
ADS Documentation

Release 2.7.2

Oracle Data Science

Dec 21, 2022

GETTING STARTED:

1	Release Notes	3
1.1	2.7.2	3
1.2	2.7.1	3
1.3	2.7.0	3
1.4	2.6.9	3
1.5	2.6.8	4
1.6	2.6.7	4
1.7	2.6.6	5
1.8	2.6.5	5
1.9	2.6.4	5
1.10	2.6.3	5
1.11	2.6.2	6
1.12	2.6.1	6
1.13	2.5.10	7
1.14	2.5.9	7
1.15	2.5.8	8
1.16	2.5.7	8
1.17	2.5.6	8
1.18	2.5.5	8
1.19	2.5.4	9
1.20	2.5.3	9
1.21	2.5.2	9
1.22	2.5.0	10
1.23	2.4.2	10
1.24	2.4.1	10
1.25	2.4.0	11
1.26	2.3.4	11
1.27	2.3.3	11
1.28	2.3.1	11
1.29	2.2.1	12
1.30	January 13, 2021	14
1.31	August 11, 2020	16
1.32	June 9, 2020	16
1.33	April 30, 2020	17
1.34	March 18, 2020	18
2	Quick Start	21
3	Installation and Setup	23
3.1	Install ADS CLI	23

3.2	Install <code>oracle-ads</code> SDK	23
3.2.1	Data Science Conda Environments	23
3.2.2	Install in Local Environments	24
3.2.2.1	Installing the <code>oracle-ads</code> base package	24
3.2.2.2	Installing extras libraries	24
4	Authentication	27
4.1	1. Authenticating Using Resource Principals	27
4.2	2. Authenticating Using API Keys	28
4.3	3. Authenticating Using Instance Principals	28
4.4	4. Overriding Defaults	28
5	CLI Configuration	31
6	Local Development Environment Setup	33
6.1	Build Development Container Image	33
6.2	Setting up Visual Studio Code	34
6.3	Working with Conda packs	34
6.3.1	create	35
6.3.2	publish	35
6.3.3	install	35
6.4	Build Your Own Container (BYOC)	35
6.4.1	Test Container image	35
6.4.2	Setup VS Code to use container as development environment	36
6.4.3	Publish image to registry	36
6.4.4	Run container image on OCI Data Science	36
7	Load Data	37
7.1	Connecting to Data Sources	37
7.1.1	Object Storage	37
7.1.2	Local Storage	38
7.1.3	Oracle Database	38
7.1.3.1	Oracle ADB to Pandas	38
7.1.3.2	Oracle Database to Pandas - No Wallet	40
7.1.3.3	Performance	41
7.1.3.4	Large Result Set	41
7.1.3.5	Very Large Result Set	42
7.1.3.6	Pandas to Oracle Database	42
7.1.4	MySQL	42
7.1.5	BDS Hive	44
7.1.5.1	Connection Parameters	44
7.1.5.2	Partition	45
7.1.5.3	Large Dataframe	46
7.1.6	HTTP(S) Sources	46
7.1.7	Convert Pandas DataFrame to <code>ADSDataset</code>	46
7.1.8	Using <code>PyArrow</code>	47
7.2	<code>DataSetFactory</code>	47
7.2.1	Connect with <code>DataSetFactory</code>	47
7.2.1.1	Object Storage	48
7.2.1.2	Local Storage	48
7.2.1.2.1	Oracle Database	49
7.2.1.3	Autonomous Database	50
7.2.1.3.1	Load from ADB	50
7.2.1.3.2	Query ADB	51
7.2.1.4	Train a Models with ADB	52

	7.2.1.5	Update ADB Tables	52
	7.2.1.6	Amazon S3	53
	7.2.1.7	HTTP(S) Sources	53
	7.2.1.8	DatasetBrowser	53
8	Label Data		55
8.1	Overview		55
8.2	Quick Start		56
8.3	Export Metadata		57
8.4	List		57
8.5	Load		57
	8.5.1	LabeledDatasetReader	58
	8.5.2	Pandas Accessor	59
8.6	Visualize		60
	8.6.1	Image	60
	8.6.2	Text	61
8.7	Examples		61
	8.7.1	Binary Text Classification	61
		8.7.1.1 Dataset	62
		8.7.1.2 Load	62
		8.7.1.3 Preprocess	62
		8.7.1.4 Train	63
		8.7.1.5 Predict	63
	8.7.2	Image Classification	63
		8.7.2.1 Data Source	63
		8.7.2.2 Load	64
		8.7.2.3 Visualize	64
		8.7.2.4 Preprocess	64
		8.7.2.5 Train	65
		8.7.2.6 Predict	65
	8.7.3	Multinomial Text Classification	65
		8.7.3.1 Dataset	65
		8.7.3.2 Load	66
		8.7.3.3 Preprocess	66
		8.7.3.4 Train	66
		8.7.3.5 Predict	67
	8.7.4	Named Entity Recognition	67
		8.7.4.1 Dataset	67
		8.7.4.2 Load	67
		8.7.4.3 Preprocess	68
		8.7.4.4 Train	69
		8.7.4.5 Predict	71
9	Transform Data		73
9.1	Loading the Dataset		73
9.2	Automated Transformations		73
9.3	Row Operations		74
	9.3.1	Delete Rows	75
	9.3.2	Reset Index	75
	9.3.3	Append Rows	75
	9.3.4	Row Filtering	75
	9.3.5	Removing Duplicated Rows	76
9.4	Column Operations		76
	9.4.1	Delete a Column	76

9.4.2	Rename a Column	76
9.4.3	Counts of Unique Values	76
9.4.4	Normalize a Column	77
9.4.5	Combine Columns	77
9.4.6	Apply a Function to a Column	77
9.4.7	Change Data Type	78
9.5	Dataset Manipulation	79
9.5.1	Categorical Encoding	79
9.5.2	One-Hot Encoding	79
9.5.3	Extract Null Values	80
9.5.4	Imputation	80
9.5.5	Combine Datasets	81
9.5.5.1	Join Datasets	81
9.5.5.2	Concatenate Datasets	82
9.6	Train/Test Datasets	82
9.7	Text Data	83
9.7.1	TextStrings	83
9.7.1.1	Overview	83
9.7.1.2	Quick Start	83
9.7.1.2.1	NLP Parse	83
9.7.1.2.2	Plugin	84
9.7.1.2.3	RegEx Match	85
9.7.1.3	NLP Parse	85
9.7.1.3.1	NLTK	85
9.7.1.3.2	spaCy	87
9.7.1.4	Plugin	89
9.7.1.4.1	Custom Plugin	89
9.7.1.4.2	OCI Language Services	90
9.7.1.5	RegEx Match	96
9.7.1.6	Still a String	97
9.7.2	Text Extraction	97
9.7.2.1	Introduction	98
9.7.2.1.1	Configure the Data Source	99
9.7.2.2	Load	99
9.7.2.2.1	Read a Dataset	99
9.7.2.2.2	Read Options	100
9.7.2.3	Augment Records	105
9.7.2.3.1	Examples	106
9.7.2.4	Custom File Processor and Backend	107
9.7.2.4.1	Custom Backend	107
9.7.2.4.2	Custom File Processor	107
9.7.2.4.3	Example	108
10	Visualize Data	109
10.1	Automatic	109
10.2	Customized	118
10.2.1	Seaborn	118
10.2.2	Matplotlib	118
10.2.3	Geographic Information System (GIS)	121
11	Train Models	123
11.1	ADSTuner	123
11.1.1	Notebook Example: Hyperparameter Optimization with ADSTuner	128
11.2	Distributed Training	136

11.2.1	Getting Started	137
11.2.1.1	Key Steps	137
11.2.1.2	Prepare container image for distributed workload	137
11.2.1.3	Check Config File generated by the Main and the Worker Nodes	137
11.2.2	Configurations	138
11.2.2.1	Networks	138
11.2.2.2	OCI Policies	138
11.2.2.3	Policy Syntax	140
11.2.3	Developer Guide	140
11.2.3.1	Build Image	140
11.2.3.2	Publish Docker Image	141
11.2.3.3	Run the container Image on the OCI Data Science or local	142
11.2.3.4	Development Flow	143
11.2.3.5	Diagnosing Infrastructure Setup	143
11.2.4	Dask	144
11.2.4.1	Creating Workloads	144
11.2.4.2	Writing Dask Code	150
11.2.4.3	Distributed XGBoost & LightGBM	151
11.2.4.3.1	LightGBM	151
11.2.4.3.2	XGBoost	152
11.2.4.4	Securing with TLS	152
11.2.4.5	Dask Cluster Tuning	155
11.2.4.5.1	Configuring dask startup options	155
11.2.4.5.2	Configuration through Environment Variables	156
11.2.4.6	Dask dashboard	157
11.2.4.6.1	Bastion Host	157
11.2.5	Horovod	158
11.2.5.1	Creating Horovod Workloads	158
11.2.5.2	Writing Distributed code with Horovod Framework	166
11.2.5.2.1	TensorFlow	166
11.2.5.2.2	PyTorch	167
11.2.5.3	Monitoring Training	173
11.2.6	PyTorch Distributed	173
11.2.6.1	Creating PyTorch Distributed Workloads	174
11.2.7	Tensorflow	184
11.2.7.1	Creating Tensorflow Workloads	184
11.2.8	Run Source Code from Git or Object Storage	197
11.2.8.1	Git Repository	197
11.2.8.2	Object Storage	198
11.2.9	YAML Schema	198
11.2.10	Troubleshooting	201
11.3	TensorBoard	201
11.3.1	Setting up local environment	202
11.3.2	Viewing logs from your experiments	202
11.3.3	Writing TensorBoard logs to Object Storage	202
11.3.3.1	PyTorch	203
11.3.3.2	TensorFlow	204
11.3.3.2.1	OCI Data Science Notebook	204
11.3.3.2.2	OCI Data Science Jobs	204
11.4	Model Evaluation	205
11.4.1	Quick Start	206
11.4.1.1	Comparing Binary Classification Models	206
11.4.1.2	Comparing Multi Classification Models	206
11.4.1.3	Comparing Regression Models	207

11.4.2	Binary Classification	208
11.4.2.1	Fairness Metrics	212
11.4.3	Multinomial Classification	213
11.4.4	Regression	220
11.5	Model Explainability	223
11.5.1	Accumulated Local Effects	224
11.5.1.1	Overview	224
11.5.1.2	Description	224
11.5.1.3	Interpretation	226
11.5.1.4	Limitations	226
11.5.1.5	Examples	227
11.5.1.6	References	230
11.5.2	Feature Dependence Explanations	230
11.5.2.1	Overview	230
11.5.2.2	Description	230
11.5.2.3	Interpretation	231
11.5.2.3.1	PDP	231
11.5.2.3.2	ICE	231
11.5.2.4	Examples	231
11.5.2.5	References	240
11.5.3	Feature Importance Explanations	240
11.5.3.1	Overview	240
11.5.3.2	Description	240
11.5.3.3	Interpretation	241
11.5.3.4	Examples	241
11.5.3.5	References	245
11.5.4	Enhanced LIME	246
11.5.4.1	Overview	246
11.5.4.2	Description	246
11.5.4.3	Interpretation	247
11.5.4.4	Example	249
11.5.4.5	References	254
11.5.5	WhatIf Explainer	254
11.5.5.1	Description	254
11.5.5.2	Example	254
12	Register and Deploy Models	259
12.1	Workflow	259
12.2	Register	260
12.2.1	Quick Start	260
12.2.1.1	Sklearn	260
12.2.1.2	XGBoost	261
12.2.1.3	LightGBM	262
12.2.1.4	PyTorch	263
12.2.1.5	Spark Pipeline	263
12.2.1.6	TensorFlow	264
12.2.1.7	Other Frameworks	265
12.2.2	Model Registration	266
12.2.2.1	Model Artifact	266
12.2.2.2	Prepare the Model Artifact	266
12.2.2.3	score.py	268
12.2.2.3.1	load_model	268
12.2.2.3.2	predict	269
12.2.2.3.3	pre_inference	269

	12.2.2.3.4	post_inference	269
	12.2.2.3.5	deserialize	269
	12.2.2.3.6	fetch_data_type_from_schema	269
	12.2.2.4	Model Introspection	269
	12.2.2.5	Save Model	270
	12.2.2.6	Save Large Model	271
	12.2.2.7	Export Model Artifact to Object Storage	271
12.2.3		Model Schema	272
	12.2.3.1	Schema Model	273
	12.2.3.2	Generating Schema	274
	12.2.3.3	Update the Schema	275
12.2.4		Model Metadata	276
	12.2.4.1	Taxonomy Metadata	276
	12.2.4.2	Custom Metadata	281
12.2.5		Customizing the Model	283
	12.2.5.1	Customize score.py	283
12.3		Deploying model	291
	12.3.1	Deploy	292
	12.3.2	Predict	293
	12.3.3	Observability	293
	12.3.4	Update Model Deployment	293
12.4		Loading model	293
	12.4.1	Load Registered Model	293
	12.4.2	Load Deployed Model	295
	12.4.3	Load Model From Object Storage	296
	12.4.4	Large Model Artifacts	297
12.5		Frameworks	298
	12.5.1	SklearnModel	298
		12.5.1.1 Overview	298
		12.5.1.2 Prepare Model Artifact	299
		12.5.1.3 Summary Status	300
		12.5.1.4 Register Model	301
		12.5.1.5 Deploy and Generate Endpoint	301
		12.5.1.6 Run Prediction against Endpoint	301
		12.5.1.7 Run Prediction with oci raw-request command	301
		12.5.1.7.1 json payload example	301
		12.5.1.7.2 numpy.ndarray payload example	302
		12.5.1.7.3 pandas.core.frame.DataFrame payload example	302
		12.5.1.7.4 dict payload example	303
		12.5.1.7.5 Expected output of raw-request command	303
	12.5.1.8	Examples	304
12.5.2		PyTorchModel	305
		12.5.2.1 Overview	305
		12.5.2.2 Prepare Model Artifact	305
		12.5.2.2.1 Save as TorchScript	305
		12.5.2.2.2 Save state_dict	306
	12.5.2.3	Verify Changes to Score.py	309
	12.5.2.4	Summary Status	309
	12.5.2.5	Register Model	310
	12.5.2.6	Deploy and Generate Endpoint	310
	12.5.2.7	Run Prediction against Endpoint	311
		12.5.2.7.1 Predict with Image	311
	12.5.2.8	Run Prediction with oci raw-request command	313
		12.5.2.8.1 torch.Tensor payload example	313

12.5.2.8.2	Expected output of raw-request command	313
12.5.2.9	Example	314
12.5.3	TensorFlowModel	315
12.5.3.1	Overview	316
12.5.3.2	Prepare Model Artifact	316
12.5.3.3	Summary Status	318
12.5.3.4	Register Model	318
12.5.3.5	Deploy and Generate Endpoint	319
12.5.3.6	Run Prediction against Endpoint	319
12.5.3.6.1	Predict with Image	319
12.5.3.7	Run Prediction with oci raw-request command	320
12.5.3.7.1	<i>numpy.ndarray</i> payload example	320
12.5.3.7.2	Expected output of raw-request command	320
12.5.3.8	Example	321
12.5.4	SparkPipelineModel	322
12.5.4.1	Overview	322
12.5.4.2	Prepare Model Artifact	323
12.5.4.3	Summary Status	325
12.5.4.4	Register Model	325
12.5.4.5	Deploy and Generate Endpoint	326
12.5.4.6	Run Prediction against Endpoint	326
12.5.4.7	Run Prediction with oci raw-request command	326
12.5.4.7.1	<i>json</i> payload example	326
12.5.4.7.2	Expected output of raw-request command	327
12.5.4.8	Example	327
12.5.5	LightGBMModel	329
12.5.5.1	Overview	329
12.5.5.2	Prepare Model Artifact	329
12.5.5.3	Summary Status	331
12.5.5.4	Register Model	332
12.5.5.5	Deploy and Generate Endpoint	332
12.5.5.6	Run Prediction against Endpoint	332
12.5.5.7	Run Prediction with oci raw-request command	333
12.5.5.7.1	<i>json</i> payload example	333
12.5.5.7.2	<i>numpy.ndarray</i> payload example	333
12.5.5.7.3	<i>pandas.core.frame.DataFrame</i> payload example	334
12.5.5.7.4	<i>dict</i> payload example	334
12.5.5.7.5	Expected output of raw-request command	334
12.5.5.8	Example	335
12.5.6	XGBoostModel	336
12.5.6.1	Overview	336
12.5.6.2	Prepare Model Artifact	337
12.5.6.3	Summary Status	338
12.5.6.4	Register Model	339
12.5.6.5	Deploy and Generate Endpoint	339
12.5.6.6	Run Prediction against Endpoint	340
12.5.6.7	Run Prediction with oci raw-request command	340
12.5.6.7.1	<i>json</i> payload example	340
12.5.6.7.2	<i>numpy.ndarray</i> payload example	340
12.5.6.7.3	<i>pandas.core.frame.DataFrame</i> payload example	341
12.5.6.7.4	<i>dict</i> payload example	341
12.5.6.7.5	Expected output of raw-request command	342
12.5.6.8	Example	342
12.5.7	AutoMLModel	343

12.5.7.1	Overview	343
12.5.7.2	Initialize	344
12.5.7.3	Summary Status	345
12.5.7.4	Example	345
12.5.8	Other Frameworks	346
12.5.8.1	Overview	346
12.5.8.2	Prepare Model Artifact	346
12.5.8.3	Summary Status	348
12.5.8.4	Example	348
13	Apache Spark	351
13.1	Quick Start	351
13.1.1	Submit a Dummy Python Script to DataFlow	351
13.1.1.1	From a Python Environment	351
13.1.1.2	From the Command Line	352
13.1.2	Real Data Flow Example with Conda Environment	353
13.1.2.1	From a Python Environment	353
13.1.2.2	From the Command Line	355
13.2	Setup and Installation	357
13.2.1	Notebook Session Development	357
13.2.1.1	Pyspark Environment	357
13.2.1.2	Configuring core-site.xml	357
13.2.2	Local Development	359
13.2.2.1	PySpark Environment	359
13.2.2.2	Developing in Visual Studio Code	359
13.2.3	Logging From DataFlow	360
13.3	Running your Spark Application on OCI Data Flow	360
13.3.1	Notebook Extension	360
13.3.2	ADS CLI	362
13.3.3	ADS Python SDK	363
13.3.3.1	YAML	369
13.4	spark-defaults.conf	372
13.4.1	odsc Command-line	372
13.4.2	Manual	373
13.4.2.1	Resource Principal	373
13.4.2.2	API Keys	373
13.5	Data Catalog Metastore	374
13.5.1	Prerequisite	374
13.5.2	Quick Start	375
13.5.2.1	Data Flow	375
13.5.2.2	Interactive Spark	377
13.5.3	Data Flow	378
13.5.3.1	PySpark Script	378
13.5.3.2	Create Application	379
13.5.3.3	Run	380
13.5.4	Interactive Spark	380
13.6	OCI Data Flow Studio	381
13.6.1	Prerequisite	382
13.6.1.1	Policies	382
13.6.2	Quick Start	382
13.6.3	Data Flow Spark Magic	383
13.6.3.1	Create Session	383
13.6.3.2	Update Session	385
13.6.3.3	Configure Session	385

13.6.3.4	Stop Session	385
13.6.3.5	Activate Session	386
13.6.3.6	Use Existing Session	386
13.6.4	Basic Spark Usage Examples	386
13.6.4.1	Spark SQL	386
13.7	[Legacy]	387
13.7.1	Prerequisite	387
13.7.2	Create a Instance	387
13.7.3	Generate a Script Using a Template	388
13.7.4	Create a Application	388
13.7.4.1	Load an Existing Application	389
13.7.4.2	Listing Applications	389
13.7.4.3	Create a Run	390
13.7.4.4	Fetching Logs	390
13.7.4.5	Edit and Synchronize PySpark Script	391
13.7.4.6	Arguments and Parameters	391
13.7.4.7	Add Third-Party Libraries	392
13.7.4.8	Fetching PySpark Output	392
13.7.5	Example Notebook: Develop Pyspark jobs locally - from local to remote workflows	394
13.7.6	Example Notebook: Using the ADB with PySpark	403
14	Big Data Service	407
14.1	Quick Start	407
14.1.1	Set Up A Conda Environment	407
14.1.2	Connect from a Notebook	407
14.1.2.1	Using the Vault	407
14.1.2.2	Without Using the Vault	408
14.2	Conda Environment	408
14.2.1	Create	408
14.2.2	Publish	408
14.3	Connect	409
14.3.1	Notebook Session	409
14.3.1.1	Using the Vault	409
14.3.1.2	Without Using the Vault	409
14.3.2	Jobs	410
14.4	File Management	410
14.4.1	FSSpec	410
14.4.1.1	Connect	410
14.4.1.2	Delete	411
14.4.1.3	Download	411
14.4.1.4	List	411
14.4.1.5	Upload	411
14.4.2	Ibis	412
14.4.2.1	Connect	412
14.4.2.2	Delete	412
14.4.2.3	Download	412
14.4.2.4	List	412
14.4.2.5	Upload	413
14.4.3	Pandas	413
14.4.3.1	Connect	413
14.4.3.2	File Handle	414
14.4.3.3	URL	414
14.4.4	PyArrow	414
14.4.4.1	Connect	415

14.4.4.2	Filesystem	415
14.5	SQL Data Management	416
14.5.1	Ibis	416
14.5.1.1	Connect	416
14.5.1.2	Query	416
14.5.1.3	Close a Connection	417
14.5.2	Impala	417
14.5.2.1	Connect	417
14.5.2.2	Create a Table	417
14.5.2.3	Query	417
14.5.2.4	Drop a Table	418
14.5.2.5	Close a Connection	418
14.5.3	PyHive	418
14.5.3.1	Connect	418
14.5.3.2	Create a Table	418
14.5.3.3	Query	419
14.5.3.4	Drop a Table	419
14.5.3.5	Close a Connection	419
15	Data Science Jobs	421
15.1	Overview	421
15.1.1	Job	422
15.1.2	Job Run	423
15.1.3	ADS Jobs	423
15.2	Data Science Job	423
15.2.1	Infrastructure	423
15.2.2	Logs	424
15.2.3	Runtime	425
15.2.4	Define a Job	426
15.2.5	Create and Run	426
15.2.6	Override Configuration	427
15.2.7	YAML Serialization	427
15.3	Run a Container	429
15.3.1	Python	429
15.3.2	YAML	430
15.4	Run a Git Repo	432
15.4.1	Python	432
15.4.2	YAML	434
15.5	Run a Notebook	436
15.5.1	Python	436
15.5.2	YAML	437
15.6	Run a Script	439
15.6.1	Python	439
15.6.2	YAML	440
15.6.3	Command Line Arguments	441
15.6.3.1	Python	441
15.6.3.2	YAML	441
15.6.4	Environment Variables	442
15.6.4.1	Python	442
15.6.4.2	YAML	443
15.7	Run Code in ZIP or Folder	445
15.7.1	ScriptRuntime	445
15.7.1.1	Python	445
15.7.1.2	YAML	446

15.7.2	PythonRuntime	446
15.7.2.1	Python	446
15.7.2.2	YAML	447
15.8	Working with OCI Data Science Jobs Using CLI	449
15.8.1	Prerequisite	449
15.8.2	Running a Pre Defined Job	450
15.8.3	Delete Job or Job Run	450
15.8.4	Cancel Job Run	450
15.8.5	Cancel Distributed Training Job	450
15.9	Monitoring With CLI	450
15.9.1	watch	450
16	Store Credentials	451
16.1	Quick Start	451
16.1.1	Auth Tokens	451
16.1.1.1	Save Credentials	451
16.1.1.2	Load Credentials	452
16.1.2	Autonomous Database	452
16.1.2.1	Save Credentials	452
16.1.2.2	Load Credentials	453
16.1.3	Big Data Service	453
16.1.3.1	Save Credentials	453
16.1.3.2	Load Credentials	454
16.1.4	MySQL	454
16.1.4.1	Save Credentials	454
16.1.4.2	Load Credentials	455
16.1.5	Oracle Database	455
16.1.5.1	Save Credentials	455
16.1.5.2	Load Credentials	456
16.2	Auth Token	456
16.2.1	Save Credentials	456
16.2.1.1	AuthTokenSecretKeeper	456
16.2.1.1.1	Save	456
16.2.1.2	Examples	457
16.2.1.2.1	Save Auth Token	457
16.2.1.2.2	Save as a yaml File	457
16.2.2	Load Credentials	457
16.2.2.1	Load	457
16.2.2.1.1	Using a with Statement	458
16.2.2.1.2	Without using a with Statement	458
16.2.2.2	Examples	458
16.2.2.2.1	Using a with Statement	458
16.2.2.2.2	Export to Environment Variables Using a with Statement	459
16.3	Autonomous Database	460
16.3.1	Save Credentials	460
16.3.1.1	ADBSecretKeeper	460
16.3.1.1.1	Save	460
16.3.1.2	Examples	461
16.3.1.2.1	Without the Wallet File	461
16.3.1.2.2	With the Wallet File	462
16.3.2	Load Credentials	462
16.3.2.1	Load	462
16.3.2.1.1	Using a with Statement	463
16.3.2.1.2	Without using a with Statement	463

16.3.2.2	Examples	463
16.3.2.2.1	Using a <code>with</code> Statement	463
16.3.2.2.2	Export to Environment Variables Using a <code>with</code> Statement	464
16.3.2.2.3	Wallet File Location	465
16.4	Big Data Service	465
16.4.1	Save Credentials	466
16.4.1.1	BDSecretKeeper	466
16.4.1.1.1	Save	466
16.4.1.2	Examples	466
16.4.1.2.1	With the Keytab and <code>kerb5</code> Config Files	466
16.4.1.2.2	Without the Keytab and <code>kerb5</code> Config Files	467
16.4.2	Load Credentials	468
16.4.2.1	Load	468
16.4.2.1.1	Using a <code>with</code> Statement	468
16.4.2.1.2	Without Using a <code>with</code> Statement	468
16.4.2.2	Examples	469
16.4.2.2.1	Using a <code>With</code> Statement	469
16.4.2.2.2	Without Using a <code>With</code> Statement	469
16.5	MySQL	470
16.5.1	Save Credentials	470
16.5.1.1	MySQLDBSecretKeeper	470
16.5.1.1.1	Save	471
16.5.1.2	Examples	471
16.5.1.2.1	Save Credentials	471
16.5.1.2.2	Save as a YAML File	472
16.5.2	Load Credentials	472
16.5.2.1	Load	472
16.5.2.1.1	Using a <code>with</code> Statement	472
16.5.2.1.2	Without Using a <code>with</code> Statement	472
16.5.2.2	Examples	472
16.5.2.2.1	Using a <code>with</code> Statement	472
16.5.2.2.2	Export the Environment Variables Using a <code>with</code> Statement	473
16.6	Oracle Database	474
16.6.1	Save Credentials	474
16.6.1.1	OracleDBSecretKeeper	474
16.6.1.2	Save	475
16.6.1.3	Examples	475
16.6.1.3.1	Save Credentials	475
16.6.1.3.2	Save as a YAML File	476
16.6.2	Load Credentials	476
16.6.2.1	Load	476
16.6.2.1.1	Using a <code>with</code> Statement	476
16.6.2.1.2	Without using a <code>with</code> Statement	476
16.6.2.2	Examples	477
16.6.2.2.1	Using a <code>with</code> Statement	477
16.6.2.2.2	Export the Environment Variable Using a <code>with</code> Statement	477
17	Model Version Set	479
17.1	Overview	479
17.2	Quick Start	479
17.3	Associate a Model	480
17.3.1	ModelVersionSet Object	481
17.3.2	Model Serialization	481
17.3.3	Context Manager	481

17.4	Create	482
17.4.1	Parameter-based Pattern	482
17.4.2	Builder Pattern	482
17.5	Delete	482
17.6	Download	483
17.6.1	.from_id()	483
17.6.2	.from_name()	483
17.7	List	483
17.7.1	ModelVersionSet	483
17.7.2	Model	483
17.8	Update	484
17.8.1	ModelVersionSet Properties	484
17.8.1.1	Parameter-based Pattern	484
17.8.1.2	Builder Pattern	484
17.8.2	Version Label	484
18	Class Documentation	487
18.1	ads package	487
18.1.1	Subpackages	487
18.1.1.1	ads.automl package	487
18.1.1.1.1	Submodules	487
18.1.1.1.2	ads.automl.driver module	487
18.1.1.1.3	ads.automl.provider module	488
18.1.1.1.4	Module contents	494
18.1.1.2	ads.catalog package	494
18.1.1.2.1	Submodules	494
18.1.1.2.2	ads.catalog.model module	494
18.1.1.2.3	ads.catalog.notebook module	501
18.1.1.2.4	ads.catalog.project module	503
18.1.1.2.5	ads.catalog.summary module	506
18.1.1.2.6	Module contents	507
18.1.1.3	ads.common package	507
18.1.1.3.1	Submodules	507
18.1.1.3.2	ads.common.card_identifier module	507
18.1.1.3.3	ads.common.auth module	507
18.1.1.3.4	ads.common.data module	516
18.1.1.3.5	ads.common.model module	518
18.1.1.3.6	ads.common.model_metadata module	521
18.1.1.3.7	ads.common.decorator.runtime_dependency module	521
18.1.1.3.8	ads.common.decorator.deprecate module	523
18.1.1.3.9	ads.common.model_introspect module	524
18.1.1.3.10	ads.common.model_export_util module	526
18.1.1.3.11	ads.common.function.fn_util module	529
18.1.1.3.12	ads.common.utils module	529
18.1.1.3.13	Module contents	538
18.1.1.3.14	ads.common.model_metadata_mixin module	538
18.1.1.4	ads.bds package	539
18.1.1.4.1	Submodules	539
18.1.1.4.2	ads.bds.auth module	539
18.1.1.4.3	Module contents	541
18.1.1.5	ads.data_labeling package	541
18.1.1.5.1	Submodules	541
18.1.1.5.2	ads.data_labeling.interface.loader module	541
18.1.1.5.3	ads.data_labeling.interface.parser module	541

18.1.1.5.4	ads.data_labeling.interface.reader module	541
18.1.1.5.5	ads.data_labeling.boundingbox module	542
18.1.1.5.6	ads.data_labeling.constants module	544
18.1.1.5.7	ads.data_labeling.data_labeling_service module	545
18.1.1.5.8	ads.data_labeling.metadata module	546
18.1.1.5.9	ads.data_labeling.ner module	548
18.1.1.5.10	ads.data_labeling.record module	549
18.1.1.5.11	ads.data_labeling.mixin.data_labeling module	550
18.1.1.5.12	ads.data_labeling.parser.export_metadata_parser module	553
18.1.1.5.13	ads.data_labeling.parser.export_record_parser module	553
18.1.1.5.14	ads.data_labeling.reader.dataset_reader module	557
18.1.1.5.15	ads.data_labeling.reader.jsonl_reader module	563
18.1.1.5.16	ads.data_labeling.reader.metadata_reader module	564
18.1.1.5.17	ads.data_labeling.reader.record_reader module	567
18.1.1.5.18	ads.data_labeling.visualizer.image_visualizer module	569
18.1.1.5.19	ads.data_labeling.visualizer.text_visualizer module	572
18.1.1.5.20	Module contents	575
18.1.1.6	ads.database package	575
18.1.1.6.1	Subpackages	575
18.1.1.6.2	Submodules	575
18.1.1.6.3	ads.database.connection module	575
18.1.1.6.4	Module contents	576
18.1.1.7	ads.dataflow package	576
18.1.1.7.1	Submodules	576
18.1.1.7.2	ads.dataflow.dataflow module	576
18.1.1.7.3	ads.dataflow.dataflowssummary module	584
18.1.1.7.4	Module contents	584
18.1.1.8	ads.dataset package	584
18.1.1.8.1	Submodules	584
18.1.1.8.2	ads.dataset.classification_dataset module	584
18.1.1.8.3	ads.dataset.correlation module	588
18.1.1.8.4	ads.dataset.correlation_plot module	588
18.1.1.8.5	ads.dataset.dask_series module	590
18.1.1.8.6	ads.dataset.dataframe_transformer module	590
18.1.1.8.7	ads.dataset.dataset module	591
18.1.1.8.8	ads.dataset.dataset_browser module	602
18.1.1.8.9	ads.dataset.dataset_with_target module	605
18.1.1.8.10	ads.dataset.exception module	610
18.1.1.8.11	ads.dataset.factory module	610
18.1.1.8.12	ads.dataset.feature_engineering_transformer module	615
18.1.1.8.13	ads.dataset.feature_selection module	615
18.1.1.8.14	ads.dataset.forecasting_dataset module	616
18.1.1.8.15	ads.dataset.helper module	616
18.1.1.8.16	ads.dataset.label_encoder module	619
18.1.1.8.17	ads.dataset.pipeline module	619
18.1.1.8.18	ads.dataset.plot module	619
18.1.1.8.19	ads.dataset.progress module	620
18.1.1.8.20	ads.dataset.recommendation module	620
18.1.1.8.21	ads.dataset.recommendation_transformer module	620
18.1.1.8.22	ads.dataset.regression_dataset module	621
18.1.1.8.23	ads.dataset.sampled_dataset module	621
18.1.1.8.24	ads.dataset.target module	622
18.1.1.8.25	ads.dataset.timeseries module	623
18.1.1.8.26	Module contents	623

18.1.1.9	ads.evaluations package	623
18.1.1.9.1	Submodules	623
18.1.1.9.2	ads.evaluations.evaluation_plot module	623
18.1.1.9.3	ads.evaluations.evaluator module	625
18.1.1.9.4	ads.evaluations.statistical_metrics module	631
18.1.1.9.5	Module contents	633
18.1.1.10	ads.explanations package	633
18.1.1.10.1	Submodules	633
18.1.1.10.2	ads.explanations.base_explainer module	633
18.1.1.10.3	ads.explanations.explainer module	633
18.1.1.10.4	ads.explanations.mlx_global_explainer module	633
18.1.1.10.5	ads.explanations.mlx_interface module	633
18.1.1.10.6	ads.explanations.mlx_local_explainer module	633
18.1.1.10.7	ads.explanations.mlx_whatif_explainer module	633
18.1.1.10.8	Module contents	633
18.1.1.11	ads.feature_engineering package	633
18.1.1.11.1	Submodules	633
18.1.1.11.2	ads.feature_engineering.exceptions module	633
18.1.1.11.3	ads.feature_engineering.feature_type_manager module	634
18.1.1.11.4	ads.feature_engineering.accessor.dataframe_accessor module	638
18.1.1.11.5	ads.feature_engineering.accessor.series_accessor module	643
18.1.1.11.6	ads.feature_engineering.accessor.mixin.correlation module	646
18.1.1.11.7	ads.feature_engineering.accessor.mixin.eda_mixin module	646
18.1.1.11.8	ads.feature_engineering.accessor.mixin.eda_mixin_series module	649
18.1.1.11.9	ads.feature_engineering.accessor.mixin.feature_types_mixin module	650
18.1.1.11.10	ads.feature_engineering.adsstring.common_regex_mixin module	652
18.1.1.11.11	ads.feature_engineering.adsstring.oci_language module	653
18.1.1.11.12	ads.feature_engineering.adsstring.string module	653
18.1.1.11.13	ads.feature_engineering.feature_type.address module	653
18.1.1.11.14	ads.feature_engineering.feature_type.base module	656
18.1.1.11.15	ads.feature_engineering.feature_type.boolean module	657
18.1.1.11.16	ads.feature_engineering.feature_type.category module	659
18.1.1.11.17	ads.feature_engineering.feature_type.constant module	662
18.1.1.11.18	ads.feature_engineering.feature_type.continuous module	664
18.1.1.11.19	ads.feature_engineering.feature_type.creditcard module	666
18.1.1.11.20	ads.feature_engineering.feature_type.datetime module	670
18.1.1.11.21	ads.feature_engineering.feature_type.discrete module	673
18.1.1.11.22	ads.feature_engineering.feature_type.document module	675
18.1.1.11.23	ads.feature_engineering.feature_type.gis module	676
18.1.1.11.24	ads.feature_engineering.feature_type.integer module	680
18.1.1.11.25	ads.feature_engineering.feature_type.ip_address module	682
18.1.1.11.26	ads.feature_engineering.feature_type.ip_address_v4 module	684
18.1.1.11.27	ads.feature_engineering.feature_type.ip_address_v6 module	686
18.1.1.11.28	ads.feature_engineering.feature_type.lat_long module	689
18.1.1.11.29	ads.feature_engineering.feature_type.object module	692
18.1.1.11.30	ads.feature_engineering.feature_type.ordinal module	693
18.1.1.11.31	ads.feature_engineering.feature_type.phone_number module	695
18.1.1.11.32	ads.feature_engineering.feature_type.string module	698
18.1.1.11.33	ads.feature_engineering.feature_type.text module	700
18.1.1.11.34	ads.feature_engineering.feature_type.unknown module	702
18.1.1.11.35	ads.feature_engineering.feature_type.zip_code module	703
18.1.1.11.36	ads.feature_engineering.feature_type.handler.feature_validator module	705
18.1.1.11.37	ads.feature_engineering.feature_type.handler.feature_warning module	710
18.1.1.11.38	ads.feature_engineering.feature_type.handler.warnings module	713

18.1.1.11.39	Module contents	714
18.1.1.12	ads.hpo package	714
18.1.1.12.1	Submodules	714
18.1.1.12.2	ads.hpo.distributions module	714
18.1.1.12.3	ads.hpo.search_cv module	717
18.1.1.12.4	ads.hpo.stopping_criterion	728
18.1.1.12.5	Module contents	729
18.1.1.13	ads.jobs package	729
18.1.1.13.1	Submodules	729
18.1.1.13.2	ads.jobs.ads_job module	729
18.1.1.13.3	ads.jobs.builders.runtimes.python_runtime module	735
18.1.1.13.4	ads.jobs.builders.infrastructure.dataflow module	744
18.1.1.13.5	ads.jobs.builders.infrastructure.dsc_job module	753
18.1.1.13.6	Module contents	763
18.1.1.14	ads.model.framework other package	763
18.1.1.14.1	Submodules	763
18.1.1.14.2	ads.model.artifact module	763
18.1.1.14.3	ads.model.generic_model module	765
18.1.1.14.4	ads.model.model_properties module	788
18.1.1.14.5	ads.model.runtime.runtime_info module	790
18.1.1.14.6	ads.model.extractor.model_info_extractor_factory module	790
18.1.1.14.7	ads.model.extractor.model_artifact module	791
18.1.1.14.8	ads.model.extractor.automl_extractor module	791
18.1.1.14.9	ads.model.extractor.xgboost_extractor module	792
18.1.1.14.10	ads.model.extractor.lightgbm_extractor module	793
18.1.1.14.11	ads.model.extractor.model_info_extractor module	794
18.1.1.14.12	ads.model.extractor.sklearn_extractor module	795
18.1.1.14.13	ads.model.extractor.keras_extractor module	797
18.1.1.14.14	ads.model.extractor.tensorflow_extractor module	798
18.1.1.14.15	ads.model.extractor.pytorch_extractor module	799
18.1.1.14.16	Module contents	800
18.1.1.15	ads.model.deployment package	800
18.1.1.15.1	Submodules	800
18.1.1.15.2	ads.model.deployment.model_deployer module	800
18.1.1.15.3	ads.model.deployment.model_deployment module	804
18.1.1.15.4	ads.model.deployment.model_deployment_properties module	809
18.1.1.15.5	Module contents	813
18.1.1.16	ads.model.framework package	813
18.1.1.16.1	Submodules	813
18.1.1.16.2	ads.model.framework.automl_model module	813
18.1.1.16.3	ads.model.framework.lightgbm_model module	818
18.1.1.16.4	ads.model.framework.pytorch_model module	823
18.1.1.16.5	ads.model.framework.sklearn_model module	828
18.1.1.16.6	ads.model.framework.tensorflow_model module	833
18.1.1.16.7	ads.model.framework.xgboost_model module	838
18.1.1.16.8	Module contents	843
18.1.1.17	ads.model.runtime package	843
18.1.1.17.1	Submodules	843
18.1.1.17.2	ads.model.runtime.env_info module	843
18.1.1.17.3	ads.model.runtime.model_deployment_details module	845
18.1.1.17.4	ads.model.runtime.model_provenance_details module	845
18.1.1.17.5	ads.model.runtime.runtime_info module	846
18.1.1.17.6	ads.model.runtime.utils module	846
18.1.1.17.7	Module contents	847

18.1.1.18	ads.oracledb package	847
18.1.1.18.1	Submodules	847
18.1.1.18.2	ads.oracledb.oracle_db module	847
18.1.1.19	ads.secrets package	847
18.1.1.19.1	Submodules	847
18.1.1.19.2	ads.secrets.secrets module	847
18.1.1.19.3	ads.secrets.adb module	851
18.1.1.19.4	ads.secrets.mysqlldb module	855
18.1.1.19.5	ads.secrets.oracledb module	857
18.1.1.19.6	ads.secrets.big_data_service module	859
18.1.1.19.7	ads.secrets.auth_token module	863
18.1.1.19.8	Module contents	865
18.1.1.20	ads.text_dataset package	865
18.1.1.20.1	Submodules	865
18.1.1.20.2	ads.text_dataset.backends module	865
18.1.1.20.3	ads.text_dataset.dataset module	867
18.1.1.20.4	ads.text_dataset.extractor module	872
18.1.1.20.5	ads.text_dataset.options module	874
18.1.1.20.6	Module contents	875
18.1.1.21	ads.vault package	875
18.1.1.21.1	Submodules	875
18.1.1.21.2	ads.vault module	875
18.1.1.21.3	Module contents	876
18.1.2	Submodules	876
18.1.3	ads.config module	876
18.1.4	Module contents	877
19	Additional Documentation	879
20	Examples	881
20.1	Load data from Object Storage	881
20.2	Load data from Autonomous DB	881
20.3	More Examples	882
21	Contributing	883
22	Security	885
23	License	887
	Python Module Index	889
	Index	893

RELEASE NOTES

1.1 2.7.2

Release date: December 20, 2022

- Fixed a bug in ADS jobs. The `job_run.watch()` method sometimes threw an exception due to an unexpected logging parameter.

1.2 2.7.1

Release date: December 14, 2022

- Fixed a bug with `ads.set_auth('resource_principal')` - <https://github.com/oracle/accelerated-data-science/issues/38>

1.3 2.7.0

Release date: December 7, 2022

- Fixed a bug in `GenericModel.prepare`. The `.model-ignore` file was not included in the `Manifest.in`.

1.4 2.6.9

Release date: December 7, 2022

- Added compatibility with Python 3.10.
- Added `update_deployment()` method to the *framework-specific* classes. Update model deployment associated with the model.
- Added `from_id()` method to the *framework-specific* classes. Load existing model by OCID directly from the OCI Models and OCI Model Deployment.
- Added `upload_artifact()` to the *framework-specific* classes. Upload model artifacts to Object Storage.
- Added `update()` method to the *framework-specific* classes. Update the model metadata for the registered model.
- Added `config`, `signer`, `signer_callable` attributes to the `ads.set_auth()` to support additional signers.
- Added support for Instance Principals authentication for the `ads opctl conda publish` and `ads opctl conda install` commands.

- Added an option for `PyTorchModel` framework allowing to serialize model in a `TorchScript` format.
- Added an option to import *framework-specific* classes directly from the `ads.model` package. Example: `from ads.model import LightGBMModel, GenericModel`.
- Fixed a bug in `ADSDataset.get_recommendations` when imbalanced correction depends on classes alpha order.
- Fixed a bug in ADS jobs. The shape configuration details were incorrectly extracted from a notebook session.
- Fixed a bug to replace the use of a deprecated API with latest API in the Model Evaluation module.

Following modules are marked as deprecated:

- `ads.catalog.model.py`.
- `ads.catalog.notebook.py`
- `ads.catalog.project.py`
- `ads.catalog.summary.py`

1.5 2.6.8

Release date: October 29, 2022

- Fixed a bug in `ads.dataset.helper` to support Python 3.8 and Python 3.9.

1.6 2.6.7

Release date: October 27, 2022

- Fixed a bug in `PyTorchModel`. The `score.py` failed when `torch.Tensor` was used as input data.
- Fixed a bug in `ads opctl conda publish` command.
- Added support for flexible shapes for Data Flow Jobs.
- Loading a model from Model Catalog (`GenericModel.from_model_catalog()`) and Model Deployment (`GenericModel.from_model_deployment()`) no longer requires a model file name.
- Switched from using `cx_Oracle` interface to the `oracledb` driver to connect to Oracle Databases.
- Added support for image attribute for the `PyTorchModel.predict()` and `TensorFlowModel.predict()` methods. Images can now be directly passed to the model Deployment predict.

The following APIs are deprecated:

- `OracleAutoMLProvider`

1.7 2.6.6

Release date: October 7, 2022

- Added `SparkPipelineModel` model serialization class for fast and easy model deployment.
- Added support for flexible shapes for Jobs and Model Deployments.
- Added support for `freeform_tags` and `defined_tags` for Model Deployments.
- Added the `populate_schema()` method to the `GenericModel` class. Populate input and output schemas for model artifacts.
- The `ADSString` was added to the Feature types system. Use the enhanced string class functionalities such as regular expression (RegEx) matching and natural language parsing within Pandas dataframes and series.
- Saving model does not require iPython dependencies

Following APIs are deprecated:

- `DatasetFactory.open`
- `ADSModel.prepare`
- `ads.common.model_export_util.prepare_generic_model`

1.8 2.6.5

Release date: September 16, 2022

- OCI SDK updated from version 2.59.0 to version 2.82.0.

1.9 2.6.4

Release date: September 14, 2022

- Added support for large models with artifact size between 2 and 6 GB. The large models can be saved to the Model Catalog, downloaded from the Model Catalog, and deployed as a Model Deployment resource.
- Added `delete()` method to the `GenericModel` class. Deletes models and associated model deployments.
- The Model Input Schema is improved to return features sorted by the `order` attribute.
- Added user-friendly default names for created Jobs, Model Deployments, and Models.

1.10 2.6.3

Release date: August 4, 2022

- Deprecated the `ads.dataflow.DataFlow` class. It has been superseded by the `ads.jobs.DataFlow` class.
- Added `prepare_save_deploy()` method to the `GenericModel` class. Prepare model artifacts and deploy the model with one command.
- Added support for binary payloads in model deployment.
- Updated `AutoMLModel`, `GenericModel`, `LightgbmModel`, `PyTorchModel`, `SklearnModel`, `TensorflowModel`, and `XgboostModel` classes to support binary payloads in model deployment.

- The maximum runtime for a Job can be limited with the `with_maximum_runtime_in_minutes()` method in the `CondaRuntime`, `DataFlowNotebookRuntime`, `DataFlowRuntime`, `GitPythonRuntime`, `NotebookRuntime`, and `ScriptRuntime` classes.
- The `ads.jobs.DataFlow` class supports Published conda environments.

1.11 2.6.2

Release date: June 21, 2022

- Added `from_model_deployment()` method to the `GenericModel` class. Now you can load a model directly from an existing model deployment.
- Moved dependencies from being default into optional installation groups:
 - `all-optional`
 - `bds`
 - `boosted`
 - `data`
 - `geo`
 - `notebook`
 - `onnx`
 - `opctl`
 - `optuna`
 - `tensorflow`
 - `text`
 - `torch`
 - `viz`

Use `python3 -m pip install "oracle-ads[XXX]"` where XXX are the group names.

1.12 2.6.1

Release date: June 1, 2022

- Added support for running a container as jobs using `ads.jobs.ContainerRuntime`.
- The `ModelArtifact` class is deprecated. Use the model serialization classes (`GenericModel`, `PyTorchModel`, `SklearnModel`, etc.).

1.13 2.5.10

Release date: May 6, 2022

- Added `BDSecretKeeper` to store and save configuration parameters to connect to Big Data service to the vault.
- Added the `krbcontext` and `refresh_ticket` functions to configure Kerberos authentication for the Big Data service.
- Added authentication options to logging APIs to allow you to pass in the OCI API key configuration or signer.
- Added the configuration file path option in the `set_auth` method. This allows you to change the path of the OCI configuration.
- Fixed a bug in AutoML for Text datasets.
- Fixed bug in `import ads.jobs` to notify users installing ADS optional dependencies.
- Fixed a bug in the generated `score.py` file, where Pandas dataframe's dtypes changed when deserializing. Now you can recover it from the input schema.
- Updated requirements to `oci>=2.59.0`.

1.14 2.5.9

Release date: April 4, 2022

- Added framework-specific model serialization to add more inputs to the generated `score.py` file.
- Added the following framework-specific classes for fast and easy model deployment:
 - `AutoMLModel`
 - `SKlearnModel`
 - `XGBoostModel`
 - `LightGBMModel`
 - `PyTorchModel`
 - `TensorFlowModel`
- Added the `GenericModel` class for frameworks not included in the preceding list:
- You can now prepare, verify, save and deploy your models using the methods in these new classes:
 - `.prepare()`: Creates `score.py`, `runtime.yaml`, and schema files for model deployment purpose, and adds the model artifacts to the model catalog.
 - `.verify()`: Helps test your model locally, before deploying it from the model catalog to an endpoint.
 - `.save()`: Saves the model and model artifacts to the model catalog.
 - `.deploy()`: Deploys a model from the model catalog to a REST endpoint.
 - `.predict()`: Calls the endpoint and creates inferences from the deployed model.
- Added support to create jobs with managed egress.
- Fixed bug in jobs, where log entries were being dropped when there were a large number of logs in a short period of time. Now you can list all logs with `jobwatch()`.

1.15 2.5.8

Release date: March 3, 2022

- Fixed bug in automatic extraction of taxonomy metadata for Sklearn models.
- Fixed bug in jobs NotebookRuntime when using non-ASCII encoding.
- Added compatibility with Python 3.8 and 3.9.
- Added an enhanced string class, called `ADSString`. It adds functionality such as regular expression (RegEx) matching, and natural language processing (NLP) parsing. The class can be expanded by registering custom plugins to perform custom string processing actions.

1.16 2.5.7

Release date: February 4, 2022

- Fixed bug in DataFlow Job creation.
- Fixed bug in ADSDataset `get_recommendations` raising `HTML is not defined` exception.
- Fixed bug in jobs ScriptRuntime causing the parent artifact folder to be zipped and uploaded instead of the specified folder.
- Fixed bug in ModelDeployment raising `TypeError` exception when updating an existing model deployment.

1.17 2.5.6

Release date: January 21, 2022

- Added support for the `storage_options` parameter in `ADSDataset.to_hdf()`.
- Fixed error message to specify `overwrite_script` or `overwrite_archive` option in `data_flow.create_app()`.
- Fixed output of multiclass evaluation plots when `ADSEvaluator()` class uses a non-default `legend_labels` option.
- Added support to connect to an Oracle Database that does not require a wallet file.
- Added support to read and write from MySQL using ADS DataFrame APIs.

1.18 2.5.5

Release date: December 9, 2021

- Fixed bug in model artifact `prepare()`, `reload()`, and `prepare_generic_model()` raising `ONNXRuntimeError` caused by the mismatched version of `skl2onnx`.

1.19 2.5.4

Release date: December 3, 2021

The following features were added:

- Added support to read exported dataset from the consolidated export file for the Data Labeling service.

Following fixes were added:

- The `DaskSeries` class was marked as deprecated.
- The `DaskSeriesAccessor` class was marked as deprecated.
- The `MLRuntime` class was marked as deprecated.
- The `ADSDataset.ddf` attribute was marked as deprecated.

1.20 2.5.3

Release date: November 29, 2021

The following features were added:

- Moved `fastavro`, `pandavro` and `openpyxl` to an optional dependency.
- Added the ability to specify the output annotation format to be `spacy` for the Entity Extraction dataset or `yolo` for the Object Detection dataset in the Data Labeling service.
- Added support to load labeled datasets from OCI Data Labeling, and return the Pandas dataframe or generator formats in the Data Labeling service.
- Added support to load labeled datasets by chunks in the Data Labeling service.

1.21 2.5.2

Release Notes: November 17, 2021

The following features were added:

- Added support to manage credentials with the OCI Vault service for ADB and Access Tokens.
- Improved model introspection functionality. The `INFERENCE_ENV_TYPE` and `INFERENCE_ENV_SLUG` parameters are no longer required.
- Updated ADS dependency requirements. Relaxed the versions for the `scikit-learn`, `scipy` and `onnx` dependencies.
- Moved `dask`, `ipywidget` and `wordcloud` to an optional dependency.
- The Boston Housing dataset was replaced with an alternative one.
- Migrated `ADSDataset` to use Pandas instead of Dask.
- Deprecated `MLRuntime`.
- Deprecated `resource_analyze` method.
- Added support for magic commands in notebooks when they run in a Job.
- Added support to download notebook and output after running it in a Job.

1.22 2.5.0

Release notes: October 20, 2021

The following features related to the Data Labeling service were added:

- Integrating with the Oracle Cloud Infrastructure Data Labeling service.
- Listing labeled datasets in the Data Labeling service.
- Exporting labeled datasets into Object Storage.
- Loading labeled datasets in the Pandas dataframe or generator formats.
- Visualizing the labeled entity extraction and object detection data.
- Converting the labeled entity extraction and object detection data to the Spacy and YOLO formats respectively.

1.23 2.4.2

The following improvements were effected:

- Improve ads import time.
- Fix the version of the *jsonschema* package.
- Update *numpy* deps to $\geq 1.19.2$ for compatibility with *TensorFlow 2.6*.
- Added progress bar when creating a Data Flow application.
- Fixed the file upload path in Data Flow.
- Added supporting tags when saving model artifacts to the model catalog.
- Updated Model Deployment authentication.
- Specify spark version in `prepare_app()` now works.
- Run a Job from a ZIP or folder.

This release has the following bug fixes:

- Fixed the default `runtime.yaml` template generated outside of a notebook session.
- Oracle DB `mixin` the batch size parameter is now passed downstream.
- `ADSModel.prepare()` and `prepare_generic_model()` `force_overwrite` deletes user-created folders.
- `prepare_generic_model` fails to create a successful artifact when taxonomy is extracted.

1.24 2.4.1

Release notes: September 27, 2021

The following dependencies were removed:

- `pyarrow`
- `python-snappy`

1.25 2.4.0

Release notes: September 22, 2021

The Data Science jobs feature is introduced and includes the following:

- Data Science jobs allow data scientists to run customized tasks outside of a notebook session.
- Running Data Science jobs and Data Flow applications through unified APIs by configuring job infrastructure and runtime parameters.
- Configuring various runtime configurations for running code from Python/Bash script, packages including multiple modules, Jupyter notebook, or a Git repository.
- Monitoring job runs and streaming log messages using the Logging service.

1.26 2.3.4

Release notes: September 20, 2021

This release has the following bug fixes:

- `prepare_generic_model` fails when used outside the Data Science notebook session
- `TextDatasetFactory` fails when used outside the Data Science notebook session

1.27 2.3.3

Release notes: September 17, 2021

- Removed dependency on `plotly`.
- `print_user_message` replaced with `logger`.

1.28 2.3.1

Release notes: August 3, 2021

This release of the model catalog includes these enhancements:

- Automatic extraction of model taxonomy metadata that lets data scientists document the use case, framework, and hyperparameters of their models.
- Improvement to the model provenance metadata, including a reference to the model training resource (notebook sessions) by passing in the `training_id` to the `.save()` method.
- Support for custom metadata which lets data scientists document the context around their models, automatic extraction references to the conda environment used to train the model, the training and validation datasets, and so on.
- Automatic extraction of the model input feature vector and prediction schemas.
- Model introspection tests that are run on the model artifact before the model is saved to the model catalog. Model introspection validates the artifact against a series of common issues and errors found with artifacts. These introspection tests are part of the model artifact code template that is included.

Feature type is an additional added module which includes the following functionality:

- Support for Exploratory Data Analysis including feature count, feature plot, feature statistics, correlation, and correlation plot.
- Support for the feature type manager that provides the tools to manage the handlers used to drive the feature type system.
- Support for the feature type validators that are a way of performing data validation and also allow a feature type to be dynamically extended so that the data validation process can be reproducible and shared across projects.
- Support for feature type warnings that allow you to automate the process of checking for data quality issues.

1.29 2.2.1

Release notes: May 7, 2021

Improvements include:

- Requires Pandas >- 1.2 and Python == 3.7.
- Upgraded the scikit-learn dependency to 0.23.2.
- Added the ADSTextDataset and the ADS Text Extraction Framework.
- Updated the ADSTuner method `.tune()` to allow asynchronous tuning, including the ability to halt, resume, and terminate tuning operations from the main process.
- Added the ability to load and save ADSTuner tuned trials to Object Storage. The tuning progress can now be saved and loaded in a different ADSTuner object.
- Added the ability to update the ADSTuner tuning search space. Hyperparameters can be changed and distribution ranges modified during tuning.
- Updated plotting functions to plot in real-time while ADSTuner asynchronous tuning operations proceed.
- Added methods to report on the remaining budget for running ADSTuner asynchronous tuner (trials and time-based budgets).
- Added a method to report the difference between the optimal and current best score for ADSTuner tuning processes with score-based stopping criteria.
- Added caching for model loading method to avoid model deserialization each time the predict method is called.
- Made the list of supported formats in `DatasetFactory.open()` more explicit.
- Moved the ADSEvaluator caption to above the table.
- Added a warning message in the `get_recommendations()` method when no recommendations can be made.
- Added a parameter in `print_summary()` to display the ranking table only.
- `list_apps` in the DataFlow class supports the optional parameter `compartment_id`.
- An exception occurs when using SVC or KNN on large datasets in `OracleAutoMLProvider`.
- Speed improvements in correlation calculations.
- Improved the name of the y-axis label in `feature_selection_trials()`.
- Automatically chooses the y-label based on the `score_metric` set in `train` if you don't set it.
- Increased the default timeout for uploading models to the model catalog.
- Improved the module documentation.
- Speed improvements in `get_recommendations()` on wide datasets.

- Speed improvements in `DatasetFactory.open()`.
- Deprecated the `frac` keyword from `DatasetFactory.open()`.
- Disabled writing `requirements.txt` when `function_artifacts = False`.
- Pretty printing of specific labels in `ADSEvaluator.metrics`.
- Removed the global setting as the only mechanism for choosing the authentication in `OCIClientFactory`.
- Added the ability to have defaults and to provide authentication information while instantiating a Provider Class.
- Added a larger time buffer for the `plot_param_importance` method.
- Migrated the `DatasetFactory` reading engine from Dask to Pandas.
- Enabling Pandas to read lists and glob of files.
- `DatasetFactory` now supports reading from Object Storage using `ocifs`.
- The `DatasetFactory` URI pattern now supports namespaces and follows the HDFS Connector format.
- The `url()` method can generate PARs for Object Storage objects.
- `DatasetFactory` now has caching for Object Storage operations.

The following issues were fixed:

- Issue with multipart upload and download in `DatasetFactory`.
- Issues with log level in `OracleAutoMLProvider`.
- Issue with `fill_value` when running `get_recommendations()`.
- Issue with an invalid training path when saving model provenance.
- Issue with errors during model deletion.
- Issues with deep copying `ADSData`.
- Evaluation plot `KeyError`.
- Dataset `show_in_notebook` issue.
- Inconsistency in preparing `ADSModels` and generic models.
- Issue with `force_overwrite` in `prepare_generic_model` not being properly triggered.
- Issue with `OracleAutoMLProvider` failing to `visualize_tuning_trials`.
- Issues with `model_prepare` trying to do feature transforms on keras and pytorch models.
- Erroneous creation of `__pycache__`.
- The `AttributeError` message when an `ApplicationSummary` or `RunSummary` object is being displayed in a notebook.
- Issues with newer versions of Dask breaking `DatasetFactory`.

AutoML is upgraded to AutoML v1.0 and the changes include:

- Switched to using Pandas Dataframes internally. AutoML now uses Pandas dataframes internally instead of Numpy dataframes, avoiding needless conversions.
- Pytorch is now an optional dependency. If Pytorch is installed, AutoML automatically considers multilayer perceptrons in its search. If Pytorch is not found, deep learning models are ignored.
- Updated the Pipeline interface to include `train()`, which runs all the pipeline stages though doesn't do the final fitting of the model (`fit()` API should be used if the final fit is needed).

- Updated the Pipeline interface to include `refit()` to allow you to refit the pipeline to an updated dataset without re-running the full pipeline again. We recommend this for advanced users only. For best results, we recommended that you rerun the full pipeline when the dataset changes.
- AutoML now reports memory usage for each trial as a part of its trial attributes. This information relies on the maximum resident size metric reported by Linux, and can sometimes be unreliable.
- `holidays` is now an optional dependency. If `holidays` is installed, AutoML automatically uses it to add `holidays` as a feature for engineering datetime columns.
- Added support for Anomaly Detection and Forecasting tasks (experimental).
- Downcast dataset to reduce pipeline training memory consumption.
- Set numpy BLAS parallelism to 1 to avoid CPU over subscription.
- Created interactive example notebooks for all supported tasks (classification, regression, anomaly detection, and forecasting), see <http://automl.oraclecorp.com/>.
- Other general bug fixes.

MLX is upgraded to MLX v1.1.1 the changes include:

- Upgrading to Python 3.7
- Upgrading to support Numpy $\geq 1.19.4$
- Upgrading to support Pandas $\geq 1.1.5$
- Upgrading to support Scikit-learn $\geq 0.23.2$
- Upgrading to support Statsmodel $\geq 0.12.1$
- Upgrading to support Dask $\geq 2.30.0$
- Upgrading to support Distributed $\geq 2.30.1$
- Upgrading to support Xgboost $\geq 1.2.1$
- Upgrading to support Category_encoders $\geq 2.2.2$
- Upgrading to support Tqdm $\geq 4.36.1$
- Fixed imputation issue when columns are all NaN.
- Fixed WhatIF internal index-reference issue.
- Fixed rare floating point problem in FD/ALE explainers.

1.30 January 13, 2021

- A full distribution of this release of ADS is found in the General Machine Learning for CPU and GPU environments. The Classic environments include the previous release of ADS.
- A distribution of ADS without AutoML and MLX is found in the remaining environments.
- `DatasetFactory` can now download files first before opening them in memory using the `.download()` method.
- Added support to archive files in creating Data Flow applications and runs.
- Support was added for loading Avro format data into ADS.
- Changed model serialization to use ONNX by default when possible on supported models.
- Added `ADSTuner`, which is a framework and model agnostic hyperparameter optimizer, use the `adstuner.ipynb` notebook for examples of how to use this feature.

- Corrected the `up_sample()` method in `get_recommendations()` so that it does not fail when all features are categorical. Up-sampling is possible for datasets containing continuous and categorical features.
- Resolved issues with serializing `ndarray` objects into JSON.
- A table of all of the ADS notebook examples can be found in our service documentation: [Oracle Cloud Infrastructure Data Science](#)
- Changed `set_documentation_mode` to false by default.
- Added unit-tests related to the dataset helper.
- Fixed the `_check_object_exists` to handle situations where the object storage bucket has more than 1000 objects.
- Added option `overwrite_script` in the `create_app()` method to allow a user to override a pre-existing file.
- Added support for newer `fsspec` versions.
- Added support for the C library `Snappy`.
- Fixed issue with uploading model provenance data due to inconsistency with OCI interface.
- Resolved issue with multiple versions of `Cryptography` being installed when installing `fbprophet`.

AutoML is upgraded to AutoML v0.5.2 and the changes include:

- AutoML is now distributed in the General Machine Learning and Data Exploration conda environments.
- Support for ONNX. AutoML models can now be serialized using ONNX by calling the `to_onnx()` API on the AutoML estimator.
- Pre-processing has been overhauled to use `sklearn` pipelines to allow serialization using ONNX. Numerical, categorical, and text columns are supported for ONNX serialization. Datetime and time series columns are not supported.
- Torch-based deep learning models, `TorchMLPClassifier` and `TorchMLPRegressor`, have been added.
- GPU support for XGBoost and torch-based models have been added. This is disabled by default and can be enabled by passing in `'gpu_id': 'auto'` in `engine_opts` in the constructor. ONNX serialization for GPUs has not been tested.
- Adaptive sampling's learning curve has been smoothened. This allows adaptive sampling to converge faster on some datasets.
- Improvements to ranking performance in feature selection were added. Feature selection is now much faster on large datasets.
- The default execution engine for AutoML has been switched to Dask. You can still use the Python multiprocessing by passing `engine='local'`, `engine_opts={'n_jobs' : -1}` to `init()`
- `GaussianNB` has been enabled in the interface by default.
- The `AdaBoostClassifier` has been disabled in the pipeline-interface by default. The ONNX converter for `AdaBoost` should not be used.
- The issue `ValueError: Found unknown categories during transform` has been fixed.
- You can manually specify a hyperparameter search space to AutoML. A new parameter was added to the pipeline. This allows you to freeze some hyperparameters or to expose further ones for tuning.
- New API: Refit an AutoML pipeline to another dataset. This is primarily used to handle updated training data, where you train the pipeline once, and refit in on newer data.
- AutoML no longer closes a user-specified Dask cluster.
- AutoML properly cleans up any existing futures on the Dask cluster at the end of fit.

MLX is upgraded to MLX v1.0.16 the changes include:

- MLX is now distributed in the General Machine Learning conda environments.
- Updated the explanation descriptions to use a base64 representation of the static plots. This obviates the need for creating a `mlx_static` directory.
- Replaced the boolean indexing in slicing Pandas dataFrame with integer indexing. After updating to Pandas `>= 1.1.0` the boolean indexing caused some issues. Integer indexing addresses these issues.
- Fixed MLX-related import warnings.
- Corrected an issue with ALE when the target values are strings.
- Removed the dependency on Paramiko.
- Addresses an issue with ALE when the target values are not of type `list`.

1.31 August 11, 2020

- Support was added to use resource principles as an authentication mechanism for ADS.
- Support was added to MLX for an additional model explanation diagnostic, Accumulated Local Effects (ALEs).
- Support was added to MLX for “What-if” scenarios in model explainability.
- Improvements were made to the correlation heatmap calculations in `show_in_notebook()`.
- Improvements were made to the model artifact.

The following bugs were fixed:

- Data Flow applications inherit the compartment assignment of the client. Runs inherit from applications by default. Compartment OCIDs can also be specified independently at the client, application, and run levels.
- The Data Flow log link for logs pulled from an application loaded into the notebook session is fixed.
- Progress bars now complete fully (in `ADSModel.prepare()` and `prepare_generic_model()`).
- `BaselineModel` is now significantly faster and can be opted out of.

MLX upgraded to MLX v1.0.10 the changes include:

- Added support to specify the `mlx_static` root path (used for ALE summary).
- Added support for making `mlx_static` directory hidden (for example, `<path>/./mlx_static/`).
- Fixed issue with the boolean features in ALE.

1.32 June 9, 2020

Numerous bug fixes including:

- Support for Data Flow applications and runs outside of a notebook session compartment. Support for specific object storage logs and script buckets at the application and run levels.
- ADS detects small shapes and gives warnings for AutoML execution.
- Removal of triggers in the Oracle Cloud Infrastructure Functions `func.yaml` file.
- `DatasetFactory.open()` incorrectly yielding a classification dataset for a continuous target was fixed.
- `LabelEncoder` producing the wrong results for category and object columns was fixed.

- An untrusted notebook issue when running model explanation visualizations were fixed.
- A warning about adaptive sampling requiring at least 1000 data points was added.
- A dtype cast float to integer into `DatasetFactory.open("csv")` was added.
- An option to specify the bucket of Data Flow logs when you create the application was added.

AutoML upgraded to 0.4.2 the changes include:

- Reduced parallelization on low compute hardware.
- Support for passing in a custom logger object in `automl.init(logger=)`.
- Support for `datetime` columns. AutoML should automatically infer `datetime` columns based on the Pandas dataframe, and perform feature engineering on them. This can also be forced by using the `col_types` argument in `pipeline.fit()`. The supported types are: `['categorical', 'numerical', 'datetime']`

MLX upgraded to MLX 1.0.7 the changes include:

- Updated the feature distributions in the PDP/ICE plots (performance improvement).
- All distributions are now shown as PMFs. Categorical features show the category frequency and continuous features are computed using a NumPy histogram (with 'auto'). They are also separate sub-plots, which are interactive.
- Classification PDP: The y-axis for continuous features is now auto-scaled (not fixed to 0-1).
- 1-feature PDP/ICE: The x-axis for continuous features now shows the entire feature distribution, whereas the plot may show a subset depending on the `partial_range` parameter (for example, `partial_range=[0.2, 0.8]` computes the PDP between the 20th and 80th percentile. The plot now shows the full distribution on the x-axis, but the line charts are only drawn between the specified percentile ranges).
- 2-feature PDP: The plot x and y axes are now auto-set to match the `partial_range` specified by the user. This ensures that the heatmap fills the entire plot by default. However, the entire feature distribution can be viewed by zooming out or clicking Autoscale in plotly.
- Support for plotting scatter plots using WebGL (`show_in_notebook(..., use_webgl=True)`) was added.
- The side issues that were causing the MLX Visualization Omitted warnings in JupyterLab were fixed.

1.33 April 30, 2020

- ADS integration with the [Oracle Cloud Infrastructure Data Flow](#) service provides a more efficient and convenient to launch a Spark application and run Spark jobs
- `show_in_notebook()` has had "head" removed from accordion and is replaced with dataset "warnings".
- `get_recommendations()` is deprecated and replaced with `suggest_recommendations()`, which returns a Pandas dataframe with all the recommendations and suggested code to implement each action.
- A progress indication of [Autonomous Data Warehouse](#) reads has been added.

AutoML updated to version 0.4.1 from 0.3.1:

- More consistent handling of stratification and random state.
- Bug-fix for LightGBM and XGBoost crashing on AMD shapes was implemented.
- Unified Proxy Models across all stages of the AutoML Pipeline, ensuring leaderboard rankings are consistent was implemented.
- Remove visual option from the interface.

- The default tuning metric for both binary and multi-class classification has been changed to `neg_log_loss`.
- Bug-fix in AutoML XGBoost, where the predicted probabilities were sometimes NaN, was implemented.
- Fixed several corner case issues in Hyperparameter Optimization.

MLX updated to version 1.0.3 from 1.0.0:

- Added support for specifying the ‘average’ parameter in `sklearn` metrics by `<metric>_<average>`, for example `F1_avg`.
- Fixed an issue with the detailed scatter plot visualizations and cutoff feature/axis names.
- Fixed an issue with the balanced sampling in the Global Feature Permutation Importance explainer.
- Updated the supported scoring metrics in MLX. The `PermutationImportance` explainer now supports a large number of classification and regression metrics. Also, many of the metrics’ names were changed.
- Updated LIME and `PermutationImportance` explainer descriptions.
- Fixed an issue where `sklearn.pipeline` wasn’t imported.
- Fixed deprecated `asscalar` warnings.

1.34 March 18, 2020

Access to ADW performance has been improved significantly

Major improvements were made to the performance of the ADW dataset loader. Your data is now loaded much faster, depending on your environment.

Change to `DatasetFactory.open()` with ADW

`DatasetFactory.open()` with `format='sql'` no longer requires the `index_col` to be specified. This was confusing, since “index” means something very different in databases. Additionally, the `table` parameter may now be either a table or a sql expression.

```
ds = DatasetFactory.open(
    connection_string,
    format = 'sql',
    table = """
        SELECT *
        FROM sh.times
        WHERE rownum <= 30
    """
)
```

No longer automatically starts an H2O cluster

ADS no longer instantiates an H2O cluster on behalf of the user. Instead, you need to `import h2o` on your own and then start your own cluster.

Profiling Dask APIs

With support for Bokeh extension, you can now profile Dask operations and visualize profiler output. For more details, see [Dask ResourceProfiler](#).

You can use the `ads.common.analyzer.resource_analyze` decorator to visualize the CPU and memory utilization of operations.

During execution, it records the following information for each timestep:

- Time in seconds since the epoch
- Memory usage in MB
- % CPU usage

Example:

```
from ads.common.analyzer import resource_analyze
from ads.dataset.dataset_browser import DatasetBrowser
@resource_analyze
def fetch_data():
    sklearn = DatasetBrowser.sklearn()
    wine_ds = sklearn.open('wine').set_target("target")
    return wine_ds
fetch_data()
```

The output shows two lines, one for the total CPU percentage used by all the workers, and one for total memory used.

Dask Upgrade

Dask is updated to version 2.10.1 with support for Oracle Cloud Infrastructure Object Storage. The 2.10.1 version provides better performance than the older version.

QUICK START

- *Install*
- *Read and Write to Object Storage, Databases and other OCI Resources*
- *OCI serverless Spark - Data Flow*
- *Evaluate Trained Models*
- *Register and Deploy Models*
- *Store and Retrieve your data source credentials*
- *Connect to existing OCI Big Data Service*

INSTALLATION AND SETUP

3.1 Install ADS CLI

Prerequisites

- Linux/Mac (Intel CPU)
- For Mac on M series - Experimental.
- For Windows: Use [Windows Subsystem for Linux \(WSL\)](#)
- python >=3.7, <3.10

ads cli provides a command line interface to Jobs API related features. Set up your development environment, build docker images compliant with Notebook session and Data Science Jobs, build and publish conda pack locally, start distributed training, etc.

Installation

Install ADS and enable CLI:

```
python3 -m pip install "oracle-ads[opctl]"
```

Tip

ads opctl subcommand lets us setup your local development environment for Data Science Jobs. More information can be found by running `ads opctl -h`

3.2 Install oracle-ads SDK

3.2.1 Data Science Conda Environments

ADS is installed in the data science conda environments. Upgrade your existing oracle-ads package by running -

```
$ python3 -m pip install oracle-ads --upgrade
```

3.2.2 Install in Local Environments

You have various options when installing ADS.

3.2.2.1 Installing the oracle-ads base package

```
$ python3 -m pip install oracle-ads
```

3.2.2.2 Installing extras libraries

The `all-optional` module will install all optional dependencies.

```
$ python3 -m pip install oracle-ads[all-optional]
```

To work with gradient boosting models, install the `boosted` module. This module includes XGBoost and LightGBM model classes.

```
$ python3 -m pip install oracle-ads[boosted]
```

For big data use cases using Oracle Big Data Service (BDS), install the `bds` module. It includes the following libraries: *ibis-framework[impala]*, *hdfs[kerberos]* and *sqlalchemy*.

```
$ python3 -m pip install oracle-ads[bds]
```

To work with a broad set of data formats (for example, Excel, Avro, etc.) install the `data` module. It includes the following libraries: *fastavro*, *openpyxl*, *pandavro*, *asteval*, *datefinder*, *htmlistparse*, and *sqlalchemy*.

```
$ python3 -m pip install oracle-ads[data]
```

To work with geospatial data install the `geo` module. It includes the *geopandas* and libraries from the *viz* module.

```
$ python3 -m pip install oracle-ads[geo]
```

Install the `notebook` module to use ADS within the Oracle Cloud Infrastructure Data Science service [Notebook Session](#). This module installs *ipywidgets* and *ipython* libraries.

```
$ python3 -m pip install oracle-ads[notebook]
```

To work with ONNX-compatible run times and libraries designed to maximize performance and model portability, install the `onnx` module. It includes the following libraries, *onnx*, *onnxruntime*, *onnxmltools*, *skl2onnx*, *xgboost*, *lightgbm* and libraries from the *viz* module.

```
$ python3 -m pip install oracle-ads[onnx]
```

For infrastructure tasks, install the `opctl` module. It includes the following libraries, *oci-cli*, *docker*, *conda-pack*, *nbconvert*, *nbformat*, and *inflection*.

```
$ python3 -m pip install oracle-ads[opctl]
```

For hyperparameter optimization tasks install the `optuna` module. It includes the *optuna* and libraries from the *viz* module.

```
$ python3 -m pip install oracle-ads[optuna]
```

For Spark tasks install the `spark` module.

```
$ python3 -m pip install oracle-ads[spark]
```

Install the `tensorflow` module to include *tensorflow* and libraries from the `viz` module.

```
$ python3 -m pip install oracle-ads[tensorflow]
```

For text related tasks, install the `text` module. This will include the *wordcloud*, *spacy* libraries.

```
$ python3 -m pip install oracle-ads[text]
```

Install the `torch` module to include *pytorch* and libraries from the `viz` module.

```
$ python3 -m pip install oracle-ads[torch]
```

Install the `viz` module to include libraries for visualization tasks. Some of the key packages are *bokeh*, *folium*, *seaborn* and related packages.

```
$ python3 -m pip install oracle-ads[viz]
```

Note

Multiple extra dependencies can be installed together. For example:

```
$ python3 -m pip install oracle-ads[notebook,viz,text]
```


AUTHENTICATION

When you are working within a notebook session, you are operating as the `datascience` Linux user. This user does not have an OCI Identity and Access Management (IAM) identity, so it has no access to the Oracle Cloud Infrastructure API. Oracle Cloud Infrastructure resources include Data Science projects, models, jobs, model deployment, and the resources of other OCI services, such as Object Storage, Functions, Vault, Data Flow, and so on. To access these resources, you must use one of the two provided authentication approaches:

4.1 1. Authenticating Using Resource Principals

Prerequisite

- You are operating within a OCI service that has resource principal based authentication configured
- You have setup the required policies allowing the `resourcetype` within which you are operating to use/manage the target OCI resources.

This is the generally preferred way to authenticate with an OCI service. A resource principal is a feature of IAM that enables resources to be authorized principal actors that can perform actions on service resources. Each resource has its own identity, and it authenticates using the certificates that are added to it. These certificates are automatically created, assigned to resources, and rotated avoiding the need for you to upload credentials to your notebook session.

Data Science enables you to authenticate using your notebook session's resource principal to access other OCI resources. When compared to using the OCI configuration and key files approach, using resource principals provides a more secure and easy way to authenticate to the OCI APIs.

You can choose to use the resource principal to authenticate while using the Accelerated Data Science (ADS) SDK by running `ads.set_auth(auth='resource_principal')` in a notebook cell. For example:

```
import ads
ads.set_auth(auth='resource_principal')
compartment_id = os.environ['NB_SESSION_COMPARTMENT_OCID']
pc = ProjectCatalog(compartment_id=compartment_id)
pc.list_projects()
```

4.2 2. Authenticating Using API Keys

Prerequisite

- You have setup api keys as per the instruction [here](#)

Use API Key setup when you are working from a local workstation or on platform which does not support resource principals.

This is the default method of authentication. You can also authenticate as your own personal IAM user by creating or uploading OCI configuration and API key files inside your notebook session environment. The OCI configuration file contains the necessary credentials to authenticate your user against the model catalog and other OCI services like Object Storage. The example notebook, *api_keys.ipynb* demonstrates how to create these files.

You can follow the steps in [api_keys.ipynb](#) for step by step instruction on setting up API Keys.

Note: If you already have an OCI configuration file (`config`) and associated keys, you can upload them directly to the `/home/datascience/.oci` directory using the JupyterLab **Upload Files** or the drag-and-drop option.

4.3 3. Authenticating Using Instance Principals

Prerequisite

- You are operating within an OCI compute instance
- You have created a Dynamic Group with Matching Rules to include your compute instances, and you have authored policies allowing this Dynamic Group to perform actions within your tenancy

For more information on Instance Principals, see [Calling Services from an Instance](#).

You can choose to use the instance principal to authenticate while using the Accelerated Data Science (ADS) SDK by running `ads.set_auth(auth='instance_principal')`. For example:

```
import ads
ads.set_auth(auth='instance_principal')
mc = ModelCatalog(compartment_id="<compartment_id>")
mc.list_models()
```

4.4 4. Overriding Defaults

The default authentication that is used by ADS is set with the `set_auth()` method. However, each relevant ADS method has an optional parameter to specify the authentication method to use. The most common use case for this is when you have different permissions in different API keys or there are differences between the permissions granted in the resource principals and your API keys.

By default, ADS uses API keys to sign requests to OCI resources. The `set_auth()` method is used to explicitly set a default signing method. This method accepts one of three strings `"api_key"`, `"resource_principal"`, or `instance_principal`.

The `~/.oci/config` configuration allow for multiple configurations to be stored in the same file. The `set_auth()` method takes is `oci_config_location` parameter that specifies the location of the configuration, and the default is `~/.oci/config`. Each configuration is called a profile, and the default profile is `DEFAULT`. The `set_auth()` method takes in a parameter `profile`. It specifies which profile in the `~/.oci/config` configuration file to use. In

this context, the profile parameter is only used when API keys are being used. If no value for profile is specified, then the DEFAULT profile section is used.

```
import ads
import oci

ads.set_auth("api_key") # default signer is set to API Keys
ads.set_auth("api_key", profile = "TEST") # default signer is set to API Keys and to use
↳TEST profile
ads.set_auth("api_key", oci_config_location = "~/test_oci/config") # default signer is
↳set to API Keys and to use non-default oci_config_location
ads.set_auth("resource_principal") # default signer is set to resource principal
↳authentication
ads.set_auth("instance_principal") # default signer is set to instance principal
↳authentication

signer = oci.auth.signers.ResourcePrincipalsFederationSigner()
ads.set_auth(config={}, signer=signer) # default signer is set to
↳ResourcePrincipalsFederationSigner

signer_callable = oci.auth.signers.ResourcePrincipalsFederationSigner
ads.set_auth(signer_callable=signer_callable) # default signer is set
↳ResourcePrincipalsFederationSigner callable
```

The auth module has helper functions that return a signer which is used for authentication. The `api_keys()` method returns a signer that uses the API keys in the `.oci` configuration directory. There are optional parameters to specify the location of the API keys and the profile section. The `resource_principal()` method returns a signer that uses resource principals. The method `default_signer()` returns either a signer for API Keys or resource principals depending on the defaults that have been set. The `set_auth()` method determines which signer type is the default. If nothing is set then API keys are the default.

Additional signers may be provided by running `set_auth()` with `signer` or `signer_callable` with optional `signer_kwargs` parameters. You can find the list of additional signers [here](#).

```
from ads.common import auth as authutil
from ads.common import oci_client as oc

# Example 1: Create Object Storage client with the default signer.
auth = authutil.default_signer()
oc.OCIClientFactory(**auth).object_storage

# Example 2: Create Object Storage client with timeout set to 6000 using resource
↳principal authentication.
auth = authutil.resource_principal({"timeout": 6000})
oc.OCIClientFactory(**auth).object_storage

# Example 3: Create Object Storage client with timeout set to 6000 using API Key
↳authentication.
auth = authutil.api_keys(oci_config="/home/datascience/.oci/config", profile="TEST",
↳kwargs={"timeout": 6000})
oc.OCIClientFactory(**auth).object_storage
```

In this example, the default authentication uses API keys specified with the `set_auth` method. However, since `os_auth` is specified to use resource principals, the notebook session uses the resource principal to access OCI Object Store.

```
set_auth("api_key") # default signer is set to api_key
os_auth = authutil.resource_principal() # use resource principal to as the preferred way_
↳to access object store
```

More signers can be created using the `create_signer()` method. With the `auth_type` parameter set to `instance_principal`, the method will return a signer that uses instance principals. For other signers there are `signer` or `signer_callable` parameters. Here are examples:

```
import ads
import oci

# Example 1. Create signer that uses instance principals
auth = ads.auth.create_signer("instance_principal")

# Example 2. Provide a ResourcePrincipalsFederationSigner object
signer = oci.auth.signers.ResourcePrincipalsFederationSigner()
auth = ads.auth.create_signer(config={}, signer=signer)

# Example 3. Create signer that uses instance principals with log requests enabled
signer_callable = oci.auth.signers.InstancePrincipalsSecurityTokenSigner
signer_kwargs = dict(log_requests=True) # will log the request url and response data_
↳when retrieving
auth = ads.auth.create_signer(signer_callable=signer_callable, signer_kwargs=signer_
↳kwargs)
```

CLI CONFIGURATION

Prerequisite

- You have completed *ADS CLI installation*

Setup default values for different options while running OCI Data Science Jobs or OCI DataFlow. By setting defaults, you can avoid inputting compartment ocid, project ocid, etc.

To setup configuration run -

```
ads opctl configure
```

This will prompt you to setup default ADS CLI configurations for each OCI profile defined in your OCI config. By default, all the files are generated in the ~/.ads_ops folder.

~/.ads_ops/config.ini will contain OCI profile defaults and conda pack related information. For example:

```
[OCI]
oci_config = ~/.oci/config
oci_profile = ANOTHERPROF

[CONDA]
conda_pack_folder = </local/path/for/saving/condapack>
conda_pack_os_prefix = oci://my-bucket@mynamespace/conda_environments/
```

~/.ads_ops/ml_job_config.ini will contain defaults for running Data Science Job. Defaults are set for each profile listed in your oci config file. Here is a sample -

```
[DEFAULT]
compartment_id = oci.xxxx.<compartment_ocid>
project_id = oci.xxxx.<project_ocid>
subnet_id = oci.xxxx.<subnet-ocid>
log_group_id = oci.xxxx.<log_group_ocid>
log_id = oci.xxxx.<log_ocid>
shape_name = VM.Standard2.2
block_storage_size_in_GB = 100

[ANOTHERPROF]
compartment_id = oci.xxxx.<compartment_ocid>
project_id = oci.xxxx.<project_ocid>
subnet_id = oci.xxxx.<subnet-ocid>
shape_name = VM.Standard2.1
log_group_id = ocid1.loggroup.oc1.xxx.xxxxx
```

(continues on next page)

(continued from previous page)

```
log_id = oci.xxxx.<log_ocid>
block_storage_size_in_GB = 50
```

~/.ads_ops/dataflow_config.ini will contain defaults for running Data Science Job. Defaults are set for each profile listed in your oci config file. Here is a sample -

```
[MYTENANCYPROF]
compartment_id = oci.xxxx.<compartment_ocid>
driver_shape = VM.Standard2.1
executor_shape = VM.Standard2.1
logs_bucket_uri = oci://mybucket@mytenancy/dataflow/logs
script_bucket = oci://mybucket@mytenancy/dataflow/mycode/
num_executors = 3
spark_version = 3.0.2
archive_bucket = oci://mybucket@mytenancy/dataflow/archive
```

LOCAL DEVELOPMENT ENVIRONMENT SETUP

Prerequisite

- You have completed *ADS CLI installation*
- You have completed *Configuration*

Setup up your workstation for development and testing your code locally before you submit it as a OCI Data Science Job. This section will guide you on how to setup environment for -

- Building an OCI Data Science compatible conda environments on your workstation or CICD pipeline and publishing to object storage
- Developing and testing code with a conda environment that is compatible with OCI Data Science Notebooks and OCI Data Science Jobs
- Developing and testing code for running Bring Your Own Container (BYOC) jobs.

Note

- In this version you cannot directly access the Service provided conda environments from ADS CLI, but you can publish a service provided conda pack from an OCI Data Science Notebook session to your object storage bucket and then use the CLI to access the published version.

6.1 Build Development Container Image

To setup an environment that matches OCI Data Science, a container image must be built. With a Data Science compatible container image you can do the following -

- Build and Publish custom conda packs that can be used within Data Science environment. Enable building conda packs in your CICD pipeline.
- Install an existing conda pack that was published from an OCI Data Science Notebook.
- Develop code locally against the same conda pack that will be used within an OCID Data Science image.

Prerequisites

1. Install docker on your workstation
2. Internet connection to pull dependencies
3. **If the access is restricted through proxy -**
 - Setup proxy environment variables `https_proxy`, `https_proxy` and `no_proxy`
 - For Linux Workstation - update proxy variables in `docker.service` file and restart docker
 - For mac - update proxy setting in the docker desktop

4. ADS cli is installed. Check CLI Installation section [here](#)

Build a container image with name `ml-job`

```
ads opctl build-image job-local
```

6.2 Setting up Visual Studio Code

[Visual Studio Code](#) can automatically run the code that you are developing inside a preconfigured container. An OCI Data Science compatible container on your workstation can be used as a development environment. Visual Studio Code can automatically launch the container using the information from `devcontainer.json`, which is created in the code directory. Automatically generate this file and further customize it with plugins. For more details [see](#)

Prerequisites

1. ADS CLI is [configured](#)
2. Install Visual Studio Code
3. [Build Development Container Image](#)
4. Install Visual Studio Code extension for [Remote Development](#)

```
ads opctl init-vscode -s <source-folder>
```

`source-folder` is a directory on your workstation where the code will reside.

`env-var` - Use this option to setup the environment variables required when the container used for development is started.

If you have to setup a proxy, you can use the following command -

```
ads opctl init-vscode -s <source-folder> --env-var http_proxy=$http_proxy https_proxy=
↪$https_proxy no_proxy=$no_proxy
```

The generated `.devcontainer.json` includes the python extension for Visual Studio Code by default.

Open the `source_folder` using Visual Studio Code. More details on running the workspace within the container can be found [here](#)

6.3 Working with Conda packs

Conda packs provide runtime dependencies and a `python` runtime for your code. The conda packs can be built inside an OCI Data Science Notebook session or you can build it locally on your workstation. `ads opctl` cli provides a way to setup a development environment to build and use the conda packs. You can push the conda packs that you build locally to Object Storage and use them in Jobs, Notebooks, Pipelines, or in Model Deployments.

Prerequisites

1. Build a local OCI Data Science Job [compatible docker image](#)
2. Connect to Object Storage through the Internet
3. Setup conda pack bucket, namespace, and authentication information using `ads opctl configure`. Refer to configuration [instructions](#).

Note

- In this version you cannot directly access the Service provided conda environments from ADS CLI, but you can publish a service provided conda pack from an OCI Data Science Notebook session to your object storage bucket and then use the CLI to access the published version.

6.3.1 create

```
ads opctl conda create -n <name> -f <path-to-environment-yaml>
```

Build conda packs from your workstation using `ads opctl conda create` subcommand.

Tip

To publish a conda pack that is natively installed on a oracle linux host (compute or laptop), use `NO_CONTAINER` environment variable to remove dependency on the ml-job container image:

```
NO_CONTAINER=1 ads opctl conda publish -s <slug> --auth <api_key/instance_principal/  
↪resource_principal>
```

6.3.2 publish

```
ads opctl conda publish -s <slug>
```

Publish conda pack to the object storage bucket from your laptop or workstation. You can use this conda pack inside OCI Data Science Service or Data Flow service.

6.3.3 install

Install conda pack using its URI. The conda pack can be used inside the docker image that you built. Use Visual Studio Code that is configured with the conda pack to help you test your code locally before submitting to OCI.

```
ads opctl conda install -u "oci://mybucket@namespace/conda_environment/path/to/my/conda"
```

6.4 Build Your Own Container (BYOC)

6.4.1 Test Container image

OCI Data Science Jobs allows you to use custom container images. `ads cli` can help you test a container image locally, publish it, and run it in OCI with a uniform interface.

Running an image locally can be conveniently achieved with “docker run” directly. “ads opctl” commands are provided here only to be symmetric to remote runs on OCI ML Job. The command looks like

```
ads opctl run -i <image-name> -e <docker entrypoint> -c "docker cmd" --env-var ENV_  
↪NAME=value -b <backend>
```

`-b` option can take either `local` - runs the container locally or `job` - runs the container on OCI.

6.4.2 Setup VS Code to use container as development environment

During the course of development, it is more productive to work within the container environment to iterate over the code. You can setup your VS Code environment to use the container as your development environment as shown here -

```
ads opctl init-vscode -i ubuntu --env-var TEST=test -v /Users/<username>/.oci:/root/.oci
```

A *devcontainer.json* is created with following contents -

```
{
  "image": "ubuntu",
  "mounts": [
    "source=/Users/<username>/.oci,target=/root/.oci,type=bind"
  ],
  "extensions": [
    "ms-python.python"
  ],
  "containerEnv": {
    "TEST": "test"
  }
}
```

6.4.3 Publish image to registry

To run a container image with OCI Data Science Job, the image needs to be in a registry accessible by OCI Data Science Job. “ads opctl publish-image” is a thin wrapper on “docker push”. The command looks like

```
ads opctl publish-image <image-name>
```

The image will be pushed to the docker registry specified in *ml_job_config.ini*. Check [configuration](#) for defaults. To overwrite the registry, use *-r <registry>*.

6.4.4 Run container image on OCI Data Science

To run a container on OCI Data Science, provide *ml_job* for *-b* option. Here is an example -

```
ads opctl run -i <region>.ocir.io/<tenancy>/ubuntu -e bash -c '-c "echo $TEST"' -b job -
↪-env-var TEST=test
```


LOAD DATA

7.1 Connecting to Data Sources

You can load data into ADS in several different ways from Oracle Cloud Infrastructure Object Storage, cx_Oracle, or S3. Following are some examples.

Begin by loading the required libraries and modules:

```
import ads
import numpy as np
import pandas as pd
from ads.common.auth import default_signer
```

7.1.1 Object Storage

To load a dataframe from Object Storage using the API keys, you can use the following example, replacing the angle bracketed content with the location and name of your file:

```
ads.set_auth(auth="api_key", oci_config_location=~/.oci/config", profile="DEFAULT")
bucket_name = <bucket-name>
file_name = <file-name>
namespace = <namespace>
df = pd.read_csv(f"oci://{bucket_name}@{namespace}/{file_name}", storage_options=default_
↪signer())
```

For a list of pandas functions to read different file format, please refer to [the Pandas documentation](#).

To load a dataframe from Object Storage using the resource principal method, you can use the following example, replacing the angle bracketed content with the location and name of your file:

```
ads.set_auth(auth='resource_principal')
bucket_name = <bucket-name>
file_name = <file-name>
namespace = <namespace>
df = pd.read_csv(f"oci://{bucket_name}@{namespace}/{file_name}", storage_options=default_
↪signer())
```

To write a pandas dataframe to object storage, provide the file name in the following format - oci://<mybucket>@<mynamespace>/<path/to/file/name>

```

ads.set_auth(auth='resource_principal')
bucket_name = <bucket-name>
file_name = <file-name>
namespace = <namespace>
df = pd.to_csv(f"oci://{bucket_name}@{namespace}/{file_name}", index=False, storage_
↳ options=default_signer())

# To setup the content type while writing to object storage, set ``oci_additional_
↳ kwargs`` attribute with ``storage_options`` to the desired content type

storage_options = default_signer()
storage_options['oci_additional_kwargs'] = {"content_type": "application/octet-stream"}
df = pd.to_csv(f"oci://{bucket_name}@{namespace}/{file_name}", index=False, storage_
↳ options=storage_options)

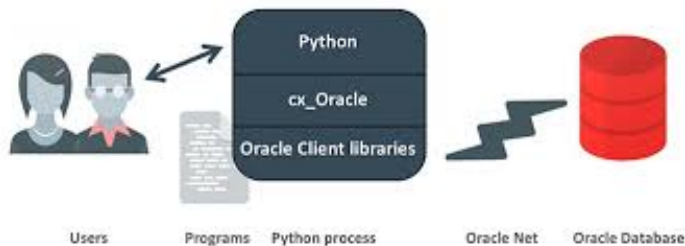
```

7.1.2 Local Storage

To load a dataframe from a local source, use functions from pandas directly:

```
df = pd.read_csv("/path/to/data.data")
```

7.1.3 Oracle Database



When using the [Oracle ADB](#) with Python the most common representation of tabular data is a [Pandas dataframe](#). When you're in a dataframe, you can perform many operations from visualization to persisting in a variety of formats.

7.1.3.1 Oracle ADB to Pandas

The Pandas `read_sql(...)` function is a general, database independent approach that uses the [SQLAlchemy - Object Relational Mapper](#) to arbitrate between specific database types and Pandas.

Read SQL query or database table into a dataframe.

This function is a convenience wrapper around `read_sql_table` and `read_sql_query` (for backward compatibility). It delegates to the specific function depending on the provided input. A SQL query is routed to `read_sql_query`, while a database table name is routed to `read_sql_table`.

Use the Pandas ADS accessor drop-in replacement, `pd.DataFrame.ads.read_sql(...)`, instead of using `pd.read_sql`.

See [how to](#) save and retrieve credentials from OCI Vault

Example using Wallet File

```

# If you are using Wallet file, provide the zip file path for `wallet_location`
connection_parameters = {
    "user_name": "<username>",
    "password": "<password>",
    "service_name": "<service_name_{high|med|low}>",
    "wallet_location": "/full/path/to/my_wallet.zip",
}
import pandas as pd
import ads

# simple read of a SQL query into a dataframe with no bind variables
df = pd.DataFrame.ads.read_sql(
    "SELECT * FROM SH.SALES",
    connection_parameters=connection_parameters,
)

# read of a SQL query into a dataframe with a bind variable. Use bind variables
# rather than string substitution to avoid the SQL injection attack vector.
df = pd.DataFrame.ads.read_sql(
    """
    SELECT
    *
    FROM
    SH.SALES
    WHERE
        ROWNUM <= :max_rows
    """,
    bind_variables={
        "max_rows" : 100
    },
    connection_parameters=connection_parameters,
)

```

Example using TLS

```

connection_parameters = {
    "user_name": "<username>",
    "password": "<password>",
    "dsn": "<connection string copied from console>",
}
import pandas as pd
import ads

# simple read of a SQL query into a dataframe with no bind variables
df = pd.DataFrame.ads.read_sql(
    "SELECT * FROM SH.SALES",
    connection_parameters=connection_parameters,
)

# read of a SQL query into a dataframe with a bind variable. Use bind variables
# rather than string substitution to avoid the SQL injection attack vector.

```

(continues on next page)

(continued from previous page)

```

df = pd.DataFrame.ads.read_sql(
    """
    SELECT
    *
    FROM
    SH.SALES
    WHERE
        ROWNUM <= :max_rows
    """,
    bind_variables={
        "max_rows" : 100
    },
    connection_parameters=connection_parameters,
)

```

7.1.3.2 Oracle Database to Pandas - No Wallet

New in version 2.5.6..

If your database connection doesn't require a wallet file, you can connect to the database by specifying host/port/sid/service name.

See [how to](#) save and retrieve credentials from OCI Vault

Example

```

connection_parameters = {
    "user_name": "<username>",
    "password": "<password>",
    "service_name": "<service_name>",
    "host": "<database hostname>",
    "port": "<database port number>"
}
import pandas as pd
import ads

# simple read of a SQL query into a dataframe with no bind variables
df = pd.DataFrame.ads.read_sql(
    "SELECT * FROM SH.SALES",
    connection_parameters=connection_parameters,
)

# read of a SQL query into a dataframe with a bind variable. Use bind variables
# rather than string substitution to avoid the SQL injection attack vector.
df = pd.DataFrame.ads.read_sql(
    """
    SELECT
    *
    FROM
    SH.SALES
    WHERE

```

(continues on next page)

(continued from previous page)

```

        ROWNUM <= :max_rows
    """
    bind_variables={
        max_rows : 100
    }
    ,
    connection_parameters=connection_parameters,
)

```

7.1.3.3 Performance

The performance is limited by three things:

- Generational latency: How long the database takes to return rows, use of indexes and writing efficient SQL mitigates this performance bottleneck.
- Network saturation: Once the network is saturated, data can't be delivered between the database and notebook environment any faster. OCI networking is very fast and this isn't usually a concern. One exception is when the network path goes over VPN or other more complex routing topologies.
- CPU latency in the notebook: Python has to collect the byte stream delivered by the database into Python data types before being promoted to Numpy objects for Pandas. Additionally, there is a cryptographic CPU overhead because the data in transit is secured with public key infrastructure (PKI).

7.1.3.4 Large Result Set

If a database query returns more rows than the memory of the client permits, you have a couple of options. The simplest is to use a larger client shape, along with increased compute performance because larger shapes come with more RAM. If that's not an option, then you can use the `pd.DataFrame.ads.read_sql` mixin in chunk mode, where the result is no longer a Pandas dataframe it is an iterator over a sequence of dataframes. You could use this read a large data set and write it to Object storage or a local file system with the following example:

```

for i, df in enumerate(pd.DataFrame.ads.read_sql(
    "SELECT * FROM SH.SALES",
    chunksize=1000000 # rows per chunk,
    connection_parameters=connection_parameters,
)):
    # each df will contain up to 1000000 rows (chunksize)
    # to write the data to object storage use oci://bucket@namespace/part_{i}.
    ↪ csv"
    df.to_csv(f"part_{i}.csv")

```

7.1.3.5 Very Large Result Set

If the data exceeds what's practical in a notebook, then the next step is to use the [Data Flow service](#) to partition the data across multiple nodes and handle data of any size up to the size of the cluster.

7.1.3.6 Pandas to Oracle Database

Typically, you would do this using `df.to_sql`. However, this uses Oracle Resource Manager to collect data and is less efficient than code that has been optimized for a specific database.

Instead, use the Pandas ADS accessor mixin.

With a `df` dataframe, writing this to the database is as simple as:

```
df.ads.to_sql(  
    "MY_TABLE",  
    connection_parameters=connection_parameters, # Should contain wallet location if you  
    ↪ are connecting to ADB  
    if_exists="replace"  
)
```

The resulting data types (if the table was created by ADS as opposed to inserting into an existing table), are governed by the following:

Pandas	Oracle
bool	NUMBER(1)
int16	INTEGER
int32	INTEGER
int64	INTEGER
float16	FLOAT
float32	FLOAT
float64	FLOAT
datetime64	TIMESTAMP
string	VARCHAR2 (Maximum length of the actual data.)

When a table is created, the length of any `VARCHAR2` column is computed from the longest string in the column. The ORM defaults to `CLOB` data, which is not correct or efficient. `CLOB`s are stored efficiently by the database, but the `c` API to query them works differently. The non-LOB columns are returned to the client through a cursor, but LOBs are handled differently resulting in an additional network fetch per row, per LOB column. ADS deals with this by creating the correct data type, and setting the correct `VARCHAR2` length.

7.1.4 MySQL

New in version 2.5.6..

To load a dataframe from a MySQL database, you must set `engine=mysql` in `pd.DataFrame.ads.read_sql`.

See [how to](#) save and retrieve credentials from OCI Vault

Example

```
connection_parameters = {  
    "user_name": "<username>",  
    "password": "<password>",
```

(continues on next page)

(continued from previous page)

```

    "host": "<database hostname>",
    "port": "<database port number>",
    "database": "<database name>"
}
import pandas as pd
import ads

# simple read of a SQL query into a dataframe with no bind variables
df = pd.DataFrame(ads.read_sql(
    "SELECT * FROM EMPLOYEE",
    connection_parameters=connection_parameters,
    engine="mysql"
))

# read of a SQL query into a dataframe with a bind variable. Use bind variables
# rather than string substitution to avoid the SQL injection attack vector.
df = pd.DataFrame(ads.read_sql(
    """
    SELECT
    *
    FROM
    EMPLOYEE
    WHERE
        emp_no <= ?
    """,
    bind_variables=(1000,)
    ,
    connection_parameters=connection_parameters,
    engine="mysql"
))

```

To save the dataframe df to MySQL, use df.ads.to_sql API with engine=mysql

```

df.ads.to_sql(
    "MY_TABLE",
    connection_parameters=connection_parameters,
    if_exists="replace",
    engine="mysql"
)

```

The resulting data types (if the table was created by ADS as opposed to inserting into an existing table), are governed by the following:

Pandas	MySQL
bool	NUMBER(1)
int16	INTEGER
int32	INTEGER
int64	INTEGER
float16	FLOAT
float32	FLOAT
float64	FLOAT
datetime64	DATETIME (Format: %Y-%m-%d %H:%M:%S)
string	VARCHAR (Maximum length of the actual data.)

7.1.5 BDS Hive

New in version 2.6.1..

To load a dataframe from BDS Hive, set `engine="hive"` in `pd.DataFrame.ads.read_sql`.

See [how to](#) save and retrieve credentials from OCI Vault

7.1.5.1 Connection Parameters

Work with BDS with Kerberos authentication

If you are working with BDS that requires Kerberos authentication, you can follow [here](#) to get connection parameters required to connect with BDS, and then follow [here](#) to save the connection parameters as well as the files needed to configure the kerberos authentication into vault. The `connection_parameters` can be set as:

```
connection_parameters = {
    "host": "<hive hostname>",
    "port": "<hive port number>",
}
```

Work with unsecure BDS

If you are working with unsecure BDS, you can set `connection_parameters` as:

```
connection_parameters = {
    "host": "<hive hostname>",
    "port": "<hive port number>",
    "auth_mechanism": "PLAIN" # for connection with unsecure BDS
}
```

Example

```
connection_parameters = {
    "host": "<database hostname>",
    "port": "<database port number>",
}
import pandas as pd
import ads

# simple read of a SQL query into a dataframe with no bind variables
df = pd.DataFrame.ads.read_sql(
```

(continues on next page)

(continued from previous page)

```

"SELECT * FROM EMPLOYEE",
connection_parameters=connection_parameters,
engine="hive"
)

# read of a SQL query into a dataframe with a bind variable. Use bind variables
# rather than string substitution to avoid the SQL injection attack vector.
df = pd.DataFrame.ads.read_sql(
    """
    SELECT
    *
    FROM
    EMPLOYEE
    WHERE
        `emp_no` <= ?
    """,
    bind_variables=(1000,)
    ,
    connection_parameters=connection_parameters,
    engine="hive"
)

```

To save the dataframe df to BDS Hive, use df.ads.to_sql API with engine="hive".

```

df.ads.to_sql(
    "MY_TABLE",
    connection_parameters=connection_parameters,
    if_exists="replace",
    engine="hive"
)

```

7.1.5.2 Partition

You can create table with partition, and then use df.ads.to_sql API with engine="hive", if_exists="append" to insert data into the table.

```

create_table_sql = f'''
    CREATE TABLE {table_name} (col1_name datatype, ...)
    partitioned by (col_name datatype, ...)
'''

df.ads.to_sql(
    "MY_TABLE",
    connection_parameters=connection_parameters,
    if_exists="append",
    engine="hive"
)

```

7.1.5.3 Large Dataframe

If the dataframe waiting to be uploaded has many rows, and the `.to_sql()` method is slow, you have other options. The simplest is to use a larger client shape, along with increased compute performance because larger shapes come with more RAM. If that's not an option, then you can follow these steps:

```
# Step1: Save your df as csv
df.to_csv(f"my_data.csv")

# Step2: Upload the csv to hdfs
hdfs_host = "<hdfs hostname>"
hdfs_port = "<hdfs port number>"
hdfs_config = {"host": hdfs_host, "port": hdfs_port, "protocol": "webhdfs"}
fs = fsspec.filesystem(**hdfs_config)
fs.upload(
    lpath="./my_data.csv",
    rpath="/user/hive/iris.csv"
)

# Step3: Create table
sql = f"""
CREATE TABLE IF NOT EXISTS {table_name} (col1_name datatype, ...)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
"""
cursor.execute(sql)

# Step4: Load data into Hive table from hdfs
hdfs_path = "./my_data.csv"
sql = f"LOAD DATA INPATH '{hdfs_path}' INTO TABLE {table_name}"
cursor.execute(sql)
```

7.1.6 HTTP(S) Sources

To load a dataframe from a remote web server source, use pandas directly and specify the URL of the data:

```
df = pd.read_csv('https://example.com/path/to/data.csv')
```

7.1.7 Convert Pandas DataFrame to ADSDataset

To convert a Pandas dataframe to ADSDataset, pass the `pandas.DataFrame` object directly into the `ADS DatasetFactory.open` method:

```
import pandas as pd
from ads.dataset.factory import DatasetFactory

df = pd.read_csv('/path/some_data.csv') # load data with Pandas

# use open...
```

(continues on next page)

(continued from previous page)

```

ds = DatasetFactory.open(df) # construct **ADS** Dataset from DataFrame

# alternative form...

ds = DatasetFactory.from_dataframe(df)

# an example using Pandas to parse data on the clipboard as a CSV and construct an ADS_
↳Dataset object
# this allows easily transferring data from an application like Microsoft Excel, Apple_
↳Numbers, etc.

ds = DatasetFactory.from_dataframe(pd.read_clipboard())

# use Pandas to query a SQL database:

from sqlalchemy import create_engine
engine = create_engine('dialect://user:pass@host:port/schema', echo=False)
df = pd.read_sql_query('SELECT * FROM mytable', engine, index_col = 'ID')
ds = DatasetFactory.from_dataframe(df)

```

7.1.8 Using PyArrow

ADS supports reading files into PyArrow dataset directly via ocifs. ocifs is installed as ADS dependencies.

```

import ocifs
import pyarrow.dataset as ds
bucket_name = <bucket_name>
namespace = <namespace>
path = <path>
fs = ocifs.OCIFFileSystem(**default_signer())
ds = ds.dataset(f"{bucket_name}@{namespace}/{path}/", filesystem=fs)

```

7.2 DataSetFactory

7.2.1 Connect with DatasetFactory

Deprecation Note

- `DataSetFactory.open` is deprecated in favor of Pandas to read from file systems.
- Pandas(>1.2.1) can connect to object storage using uri format - `oci://bucket@namespace/path/to/data`.
- To read from Oracle database or MySQL, see DataBase sections under [Connecting to Datasources](#)
- `DataSetFactory.from_dataframe` is supported to create `ADSDataset` class from pandas dataframe

See [Connecting to Datasources](#) for examples.

You can load data into ADS in several different ways from Oracle Cloud Infrastructure Object Storage, cx_Oracle, or S3. Following are some examples.

Begin by loading the required libraries and modules:

```
import ads
import numpy as np
import pandas as pd

from ads.dataset.dataset_browser import DatasetBrowser
from ads.dataset.factory import DatasetFactory
```

7.2.1.1 Object Storage

To open a dataset from Object Storage using the resource principal method, you can use the following example, replacing the angle bracketed content with the location and name of your file:

```
import ads
import os

from ads.dataset.factory import DatasetFactory

ads.set_auth(auth='resource_principal')
bucket_name = <bucket-name>
file_name = <file-name>
namespace = <namespace>
storage_options = {'config': {}, 'tenancy': os.environ['TENANCY_OCID'], 'region': os.
    ↪environ['NB_REGION']}
ds = DatasetFactory.open(f"oci://{bucket_name}@{namespace}/{file_name}", storage_
    ↪options=storage_options)
```

To open a dataset from Object Storage using the Oracle Cloud Infrastructure configuration file method, include the location of the file using this format `oci://<bucket_name>@<namespace>/<file_name>` and modify the optional parameter `storage_options`. Insert:

- The path to your [Oracle Cloud Infrastructure configuration file](#),
- The profile name you want to use.

For example:

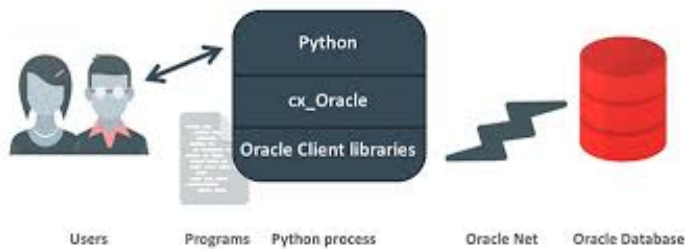
```
ds = DatasetFactory.open("oci://<bucket_name>@<namespace>/<file_name>", storage_options_
    ↪= {
    "config": "~/.oci/config",
    "profile": "DEFAULT"
})
```

7.2.1.2 Local Storage

To open a dataset from a local source, use `DatasetFactory.open` and specify the path of the data file:

```
ds = DatasetFactory.open("/path/to/data.data", format='csv', delimiter=" ")
```

7.2.1.2.1 Oracle Database



To connect to Oracle Databases from Python, you use the `cx_Oracle` package that conforms to the Python database API specification.

You must have the client credentials and connection information to connect to the database. The client credentials include the wallet, which is required for all types of connections. Use these steps to work with ADB and wallet files:

1. From the Console, go to the Oracle Cloud Infrastructure ADW or ATP instance page that you want to load the dataset from, and then click **DB Connection**.
2. Click **Download Wallet**.
3. You have to enter a password. This password is used for some ADB connections, but not the ones that are used in the notebook.
4. Create a folder for your wallet in the notebook environment (`<path_to_wallet_folder>`).
5. Upload your wallet files into `<path_to_wallet_folder>` folder using the Jupyterlab Upload Files button.
6. Open the `sqlnet.ora` file from the wallet files, and then configure the `METHOD_DATA` to be: `METHOD_DATA = (DIRECTORY="<path_to_wallet_folder>")`
7. Set the env variable, `TNS_ADMIN`. `TNS_ADMIN`, to point to the wallet you want to use.

In this example a Python dictionary, `creds` is used to store the credentials. However, it is poor security practice to store this information in a notebook. The notebook `ads-examples/ADB_working_with.ipynb` gives an example of how to store them in Block Storage.

```
creds = {}
creds['tns_admin'] = <path_to_wallet_folder>
creds['sid'] = <your SID>
creds['user'] = <database username>
creds['password'] = <database password>
```

Once your Oracle client is setup, you can use `cx_Oracle` directly with Pandas as in this example:

```
import pandas as pd
import cx_Oracle
import os

os.environ['TNS_ADMIN'] = creds['tns_admin']
with cx_Oracle.connect(creds['user'], creds['password'], creds['sid']) as ora_conn:
    df = pd.read_sql(''
        SELECT ename, dname, job, empno, hiredate, loc
        FROM emp, dept
        WHERE emp.deptno = dept.deptno
```

(continues on next page)

(continued from previous page)

```
ORDER BY ename
'', con=ora_conn)
```

You can also use `cx_Oracle` within ADS by creating a connection string:

```
os.environ['TNS_ADMIN'] = creds['tns_admin']
from ads.dataset.factory import DatasetFactory
uri = 'oracle+cx_Oracle://' + creds['user'] + ':' + creds['password'] + '@' + creds['sid']
ds = DatasetFactory.open(uri, format="sql", table=table, index_col=index_col)
```

7.2.1.3 Autonomous Database



Oracle has two configurations of Autonomous Databases. They are the Autonomous Data Warehouse (ADW) and the Autonomous Transaction Processing (ATP) database. Both are fully autonomous databases that scale elastically, deliver fast query performance, and require minimal database administration.

Note: To access [ADW](#), review the Autonomous Database configuration section. It shows you how to get the client credentials (wallet) and set up the proper environment variable.

7.2.1.3.1 Load from ADB

After you have stored the ADB username, password, and database name (SID) as variables, you can build the URI as your connection source.

```
uri = 'oracle+cx_Oracle://' + creds['user'] + ':' + creds['password'] + '@' + creds['sid']
```

You can use ADS to query a table from your database, and then load that table as an `ADSDataset` object through `DatasetFactory`. When you open `DatasetFactory`, specify the name of the table you want to pull using the `table` variable for a given table. For SQL expressions, use the `table` parameter also. For example, (`table="SELECT * FROM sh.times WHERE rownum <= 30"`).

```
os.environ['TNS_ADMIN'] = creds['tns_admin']
ds = DatasetFactory.open(uri, format="sql", table=table, target='label')
```

7.2.1.3.2 Query ADB

- Query using Pandas

This example shows you how to query data using Pandas and `sqlalchemy` to read data from ADB:

```
from sqlalchemy import create_engine
import os

os.environ['TNS_ADMIN'] = creds['tns_admin']
engine = create_engine(uri)
df = pd.read_sql('SELECT * from <TABLENAME>', con=engine)
```

You can convert the `pd.DataFrame` into `ADSDataset` using the `DatasetFactory.from_dataframe()` function.

```
ds = DatasetFactory.from_dataframe(df)
```

These two examples run a simple query on ADW data. With `read_sql_query` you can use SQL expressions not just for tables, but also to limit the number of rows and to apply conditions with filters, such as (`where`).

```
ds = pd.read_sql_query('SELECT * from <TABLENAME>', uri)
```

```
ds = pd.read_sql_query('SELECT * FROM emp WHERE ROWNUM <= 5', uri)
```

- Query using `cx_Oracle`

You can also query data from ADW using `cx_Oracle`. Use the `cx_Oracle` 7.0.0 version with ADS. Ensure that you change the dummy `<TABLENAME>` placeholder to the actual table name you want to query data from, and the dummy `<COLNAME>` placeholder to the column name that you want to select:

```
import
import pandas as pd
import numpy as np
import os

os.environ['TNS_ADMIN'] = creds['tns_admin']
connection = cx_Oracle.connect(creds['user'], creds['password'], creds['sid'])
cursor = connection.cursor()
results = cursor.execute("SELECT * from <TABLENAME>")

data = results.fetchall()
df = pd.DataFrame(np.array(data))

ds = DatasetFactory.from_dataframe(df)
```

```
results = cursor.execute('SELECT <COLNAME> from <TABLENAME>').fetchall()
```

Close the cursor and connection using the `.close()` method:

```
cursor.close()
connection.close()
```

7.2.1.4 Train a Models with ADB

After you load your data from ADB, the `ADSDataset` object is created, which allows you to build models using AutoML.

```
from ads.automl.driver import AutoML
from ads.automl.provider import OracleAutoMLProvider

train, test = ds.train_test_split()
model, baseline = AutoML(train, provider= OracleAutoMLProvider()).train(model_list=[
    ↪ "LGBMClassifier"])
```

7.2.1.5 Update ADB Tables

To add predictions to a table, you can either update an existing table, or create a new table with the added predictions. There are many ways to do this. One way is to use the model to update a CSV file, and then use Oracle SQL*Loader or SQL*Plus.

This example adds predictions programmatically using `cx_Oracle`. It uses `executemany` to insert rows as tuples created using the model's `predict` method:

```
ds = DatasetFactory.open("iris.csv")

create_table = '''CREATE TABLE IRIS_PREDICTED (,
                sepal_length number,
                sepal_width number,
                petal_length number,
                petal_width number,
                SPECIES VARCHAR2(20),
                yhat VARCHAR2(20),
                )'''

connection = cx_Oracle.connect(creds['user'], creds['password'], creds['sid'])
cursor = connection.cursor()
cursor.execute(create_table)

ds_res.to_sql('predicted_iris', con=engine, index=False, if_exists="append")\

rows = [tuple(x) for x in ds_res.values]

cursor.executemany("""
    insert into IRIS_PREDICTED
        (sepal_length, sepal_width, petal_length, petal_width, SPECIES, yhat)
    values (:1, :2, :3, :4, :5, :6)""",
    rows
)

connection.commit()
cursor.close()
connection.close()
```

For some models, you could also use `predict_proba` to get an array of predictions and their confidence probability.

7.2.1.6 Amazon S3

You can open Amazon S3 public or private files in ADS. For private files, you must pass the right credentials through the ADS `storage_options` dictionary. If you have large S3 files, then you benefit from an increased blocksize.

```
ds = DatasetFactory.open("s3://bucket_name/iris.csv", storage_options = {
    'key': 'aws key',
    'secret': 'aws secret',
    'blocksize': 10000000,
    'client_kwargs': {
        "endpoint_url": "https://s3-us-west-1.amazonaws.com"
    }
})
```

7.2.1.7 HTTP(S) Sources

To open a dataset from a remote web server source, use `DatasetFactory.open()` and specify the URL of the data:

```
ds = DatasetFactory.open('https://example.com/path/to/data.csv', target='label')
```

7.2.1.8 DatasetBrowser

`DatasetBrowser` allows easy access to datasets from reference libraries and index websites, such as scikit-learn. To see the supported libraries, use the `list()` function:

```
DatasetBrowser.list()
```

```
['web', 'sklearn', 'seaborn', 'R']
```

To see which dataset is available from scikit-learn, use:

```
sklearn = DatasetBrowser.sklearn()
sklearn.list()
```

```
['boston', 'breast_cancer', 'diabetes', 'iris', 'wine', 'digits']
```

Datasets are provided as a convenience. Datasets are considered Third Party Content and are not considered Materials under Your agreement with Oracle applicable to the Services. Review the [dataset license](#).

To explore one of the datasets, use `open()` specifying the name of the dataset:

```
ds = sklearn.open('wine')
```


LABEL DATA

The Oracle Cloud Infrastructure (OCI) Data Labeling service allows you to create and browse datasets, view data records (text, images) and apply labels for the purposes of building AI/machine learning (ML) models. The service also provides interactive user interfaces that enable the labeling process. After you label records, you can export the dataset as line-delimited JSON Lines (JSONL) for use in model development.

Datasets are the core resource available within the Data Labeling service. They contain records and their associated labels. A record represents a single image or text document. Records are stored by reference to their original source such as path on Object Storage. You can also upload records from local storage. Labels are annotations that describe a data record. There are three different dataset formats, each having its respective annotation classes:

- Images: Single label, multiple label, and object detection. Supported image types are `.png`, `.jpeg`, and `.jpg`.
- Text: Single label, multiple label, and entity extraction. Plain text, `.txt`, files are supported.
- Document: Single label and multiple label. Supported document types are `.pdf` and `.tiff`.

8.1 Overview

The Oracle Cloud Infrastructure (OCI) Data Labeling service allows you to create and browse datasets, view data records (text, images) and apply labels for the purposes of building AI/machine learning (ML) models. The service also provides interactive user interfaces that enable the labeling process. After you label records, you can export the dataset as line-delimited JSON Lines (JSONL) for use in model development.

Datasets are the core resource available within the Data Labeling service. They contain records and their associated labels. A record represents a single image or text document. Records are stored by reference to their original source such as path on Object Storage. You can also upload records from local storage. Labels are annotations that describe a data record. There are three different dataset formats, each having its respective annotation classes:

- Images: Single label, multiple label, and object detection. Supported image types are `.png`, `.jpeg`, and `.jpg`.
- Text: Single label, multiple label, and entity extraction. Plain text, `.txt`, files are supported.
- Document: Single label and multiple label. Supported document types are `.pdf` and `.tiff`.

8.2 Quick Start

The following examples provide an overview of how to use ADS to work with the Data Labeling service.

List all the datasets in the compartment:

```
from ads.data_labeling import DataLabeling
dls = DataLabeling()
dls.list_dataset()
```

With a labeled data set, the details of the labeling is called the export. To generate the export and get the path to the metadata JSONL file, you can use `export()` with these parameters:

- `dataset_id`: The OCID of the Data Labeling dataset to take a snapshot of.
- `path`: The Object Storage path to store the generated snapshot.

```
metadata_path = dls.export(
    dataset_id="<dataset_id>",
    path="oci://<bucket_name>@<namespace>/<prefix>"
)
```

To load the labeled data into a Pandas dataframe, you can use `LabeledDatasetReader` object that has these parameters:

- `materialize`: Load the contents of the dataset. This can be quite large. The default is *False*.
- `path`: The metadata file path that can be local or object storage path.

```
from ads.data_labeling import LabeledDatasetReader
ds_reader = LabeledDatasetReader.from_export(
    path="<metadata_path>",
    materialize=True
)
df = ds_reader.read()
```

You can also read labeled datasets from the OCI Data Labeling Service into a Pandas dataframe using `LabeledDatasetReader` object by specifying `dataset_id`:

```
from ads.data_labeling import LabeledDatasetReader
ds_reader = LabeledDatasetReader.from_DLS(
    dataset_id="<dataset_ocid>",
    materialize=True
)
df = ds_reader.read()
```

Alternatively, you can use the `.read_labeled_data()` method by either specifying `path` or `dataset_id`.

This example loads a labeled dataset and returns a Pandas dataframe containing the content and the annotations:

```
df = pd.DataFrame.ads.read_labeled_data(
    path="<metadata_path>",
    materialize=True
)
```

The following example loads a labeled dataset from the OCI Data Labeling, and returns a Pandas dataframe containing the content and the annotations:

```
df = pd.DataFrame.ads.read_labeled_data(
    dataset_id="<dataset_ocid>",
    materialize=True
)
```

8.3 Export Metadata

To obtain a handle to a `DataLabeling` object, you call the `DataLabeling()` constructor. The default compartment is the same compartment as the notebook session, but the `compartment_id` parameter can be used to select a different compartment.

To work with the labeled data, you need a snapshot of the dataset. The `export()` method copies the labeled data from the Data Labeling service into a bucket in Object Storage. The `.export()` method has the following parameters:

- `dataset_id`: The OCID of the Data Labeling dataset to take a snapshot of.
- `path`: The Object Storage path to store the generated snapshot.

The export process creates a JSONL file that contains metadata about the labeled dataset in the specified bucket. There is also a record JSONL file that stores the image, text, or document file path of each record and its label.

The `export()` method returns the path to the metadata file that was created in the export operation.

```
from ads.data_labeling import DataLabeling
dls = DataLabeling()
metadata_path = dls.export(
    dataset_id="<dataset_id>",
    path="oci://<bucket_name>@<namespace>/<prefix>"
)
```

8.4 List

The `.list_dataset()` method generates a list of the available labeled datasets in the compartment. The compartment is set when you call `DataLabeling()`. The `.list_dataset()` method returns a Pandas dataframe where each row is a dataset.

```
from ads.data_labeling import DataLabeling
dls = DataLabeling(compartment_id="<compartment_id>")
dls.list_dataset()
```

8.5 Load

The returned value from the `.export()` method is used to load a dataset. You can load a dataset into a Pandas dataframe using `LabeledDatasetReader` or a Pandas accessor. The `LabeledDatasetReader` creates an object that allows you to perform operations, such as getting information about the dataset without having to load the entire dataset. It also allows you to read the data directly into a Pandas dataframe or to use an iterator to process the records one at a time. The Pandas accessor approach provides a convenient method to load the data in a single command.

8.5.1 LabeledDatasetReader

Call the `.from_export()` method on `LabeledDatasetReader` to construct an object that allows you to read the data. You need the metadata path that was generated by the `.export()` method. Optionally, you can set `materialize` to `True` to load the contents of the dataset. It's set to `False` by default.

```
from ads.data_labeling import LabeledDatasetReader
ds_reader = LabeledDatasetReader.from_export(
    path=metadata_path,
    materialize=True
)
```

You can explore the metadata information of the dataset by calling `info()` on the `LabeledDatasetReader` object. You can also convert the metadata object to a dictionary using `to_dict`:

```
metadata = ds_reader.info()
metadata.labels
metadata.to_dict()
```

On the `LabeledDatasetReader` object, you call `read()` to load the labeled dataset. By default, it's read into a Pandas dataframe. You can specify the output annotation format to be `spacy` for the Entity Extraction dataset or `yolo` for the Object Detection dataset.

An Entity Extraction dataset is a dataset type that supports natural language processing named entity recognition (NLP NER). [Here is an example of spacy format.](#) A Object Detection dataset is a dataset type that contains data from detecting instances of objects of a certain class within an image. [Here is an example of yolo format.](#)

```
df = ds_reader.read()
df = ds_reader.read(format="spacy")
df = ds_reader.read(format="yolo")
```

When a dataset is too large, you can read it in small portions. The result is presented as a generator.

```
for df in ds_reader.read(chunksize=10):
    df.head()
```

Alternatively, you can call `read(iterator=True)` to return a generator of the loaded dataset, and loop all the records in the `ds_generator` by running:

```
ds_generator = ds_reader.read(iterator=True)
for item in ds_generator:
    print(item)
```

The `iterator` parameter can be combined with the `chunksize` parameter. When you use the two parameters, the result is also presented as a generator. Every item in the generator is a list of dataset records.

```
for items in ds_reader.read(iterator=True, chunksize=10):
    print(items)
```

8.5.2 Pandas Accessor

The Pandas accessor approach allows you to read a labeled dataset into a Pandas dataframe using a single command.

Use the `.read_labeled_data()` method to read the metadata file, record file, and all the corpus documents. To do this, you must know the metadata path that was created from the `.export()` method. Optionally you can set `materialize` to `True` to load content of the dataset. It's set to `False` by default. The `read_labeled_data()` method returns a dataframe that is easy to work with.

This example loads a labeled dataset and returns a Pandas dataframe containing the content and the annotations:

```
import pandas as pd
df = pd.DataFrame.ads.read_labeled_data(
    path="<metadata_path>",
    materialize=True
)
```

If you'd like to load a labeled dataset from the OCI Data Labeling, you can specify the `dataset_id`, which is dataset OCID that you'd like to read.

The following example loads a labeled dataset from the OCI Data Labeling and returns a Pandas dataframe containing the content and the annotations:

```
import pandas as pd
df = pd.DataFrame.ads.read_labeled_data(
    dataset_id="<dataset_ocid>",
    materialize=True
)
```

You can specify the output annotation format to be `spacy` for the Entity Extraction dataset or `yolo` for the Object Detection dataset.

```
import pandas as pd
df = pd.DataFrame.ads.read_labeled_data(
    dataset_id="<dataset_ocid>",
    materialize=True,
    format="spacy"
)
```

An example of a dataframe loaded with the labeled dataset is:

	Path	Content	Annotations
0	oci://hosted-ds-datasets@bigdatadatasciencelar...	From: luriem@alleg.edu(Michael Lurie) The Libe...	0
1	oci://hosted-ds-datasets@bigdatadatasciencelar...	From: nsmca@aurora.alaska.edu\nSubject: 30826\...	1
2	oci://hosted-ds-datasets@bigdatadatasciencelar...	From: aws@iti.org (Allen W. Sherzer)\nSubject:...	1
3	oci://hosted-ds-datasets@bigdatadatasciencelar...	Subject: Re: quick way to tell if your local b...	0
4	oci://hosted-ds-datasets@bigdatadatasciencelar...	Subject: Best Sportwriters...\nFrom: csc2imd@c...	0

8.6 Visualize

After the labeled dataset is loaded in a Pandas dataframe, you can visualize it using ADS. The visualization functionality only works if there are no transformations made to the *Annotations* column.

8.6.1 Image

An image dataset, with an Object Detection annotation class, can have selected image records visualized by calling the `.render_bounding_box()` method. You can provide customized colors for each label. If the `path` parameter is specified, the annotated image file is saved to that path. Otherwise, the image is displayed in the notebook session. The maximum number of records to display is set to 50 by default. This setting can be changed with the `limit` parameter:

```
df.head(1).ads.render_bounding_box()    # without user defined colors

df.iloc[1:3,:].ads.render_bounding_box(
    options={"default_color": "white",
            "colors": {"flower": "orange", "temple": "green"}},
    path="test.png"
)
```

An example of a single labeled image record is similar to:



Optionally, you can convert the bounding box to YOLO format by calling `to_yolo()` on bounding box. The labels are mapped to the index value of each label in the `metadata.labels` list.

```
df["Annotations"] = df.Annotations.apply(
    lambda items: [item.to_yolo(metadata.labels) for item in items] if items else None
)
```


8.6.2 Text

For a text dataset, with an entity extraction annotation class, you can also visualize selected text records by calling `.render_ner()`, and optionally providing customized colors for each label. By default, a maximum of 50 records are displayed. However, you can adjust this using the `limit` parameter:

```
df.head(1).ads.render_ner() # without user defined colors

df.iloc[1:3,:].ads.render_ner(options={"default_color": "#DDEECC",
                                       "colors": {"company": "#DDEECC",
                                                  "person": "#FFAAAA",
                                                  "city": "#CCC"}})
```

This is an example output for a single labeled text record:

COFFEE, SUGAR AND COCOA EXCHANGE NAMES CHAIRMAN The New York CITY Coffee, Sugar and Cocoa Exchange (CSCE COMPANY) elected former first vice chairman Gerald PERSON Clancy to a two-year term as chairman of the board of managers, replacing previous chairman Howard Katz PERSON . Katz PERSON , chairman since 1985, will remain a board member. Clancy PERSON currently serves on the Exchange board of managers as chairman of its appeals, executive, pension and political action committees. The CSCE COMPANY also elected Charles Nastro PERSON , executive vice president of Shearson Lehman Bros COMPANY , as first vice chairman. Anthony Maccia PERSON , vice president of Woodhouse COMPANY , Drake PERSON and Carey PERSON , was named second vice chairman, and Clifford Evans PERSON , president of Demico Futures PERSON , was elected treasurer.

Optionally, you can convert the entities by calling `to_spacy()`:

```
df["Annotations"] = df.Annotations.apply(
    lambda items: [item.to_spacy() for item in items] if items else None
)
```

8.7 Examples

8.7.1 Binary Text Classification

This example will demonstrate how to do binary text classification. It will demonstrate a typical data science workflow using a single label dataset from the Data Labeling Service (DLS).

Start by loading in the required libraries:

```
import ads
import oci
import os
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
```

8.7.1.1 Dataset

A subset of the 20 Newsgroups dataset is used in this example. The complete dataset is a collection of approximately 20,000 newsgroup documents partitioned across 20 different newsgroups. The dataset is popular for experiments where the machine learning application predicts which newsgroup a record belongs to.

Since this example is a binary classification, only the `rec.sport.baseball` and `sci.space` newsgroups are used. The dataset was previously labeled in the Data Labeling service. The metadata was exported and saved in a publicly accessible Object Storage bucket.

The data was previously labeled in the Data Labeling service. The metadata was exported and was saved in a publicly accessible Object Storage bucket. The metadata JSONL file is used to import the data and labels.

8.7.1.2 Load

You use the `.read_labeled_data()` method to read in the metadata file, record file, and the entire corpus of documents. Only the metadata file has to be specified because it contains references to the record and corpus documents. The `.read_labeled_data()` method returns a dataframe that is easy to work with.

The next example loads a labeled dataset, and returns the text from each email and the labeled annotation:

```
df = pd.DataFrame.ads.read_labeled_data(
    "oci://hosted-ds-datasets@bigdatadatasciencelarge/DLS/text_single_label_20news/
    ↪ metadata.jsonl",
    materialize=True
)
```

8.7.1.3 Preprocess

The data needs to be standardized. The next example performs the following operations:

- Converts the text to lower case.
- Uses a regular expression (RegEx) command to remove any character that is not alphanumeric, underscore, or whitespace.
- Replace the sequence of characters `\n` with a space.

The binary classifier model you train is a decision tree where the features are based on n-grams of the words. You use n-grams that are one, two, and three words long (unigrams, bigrams, and trigrams). The vectorizer removes English stop words because they provide little value to the model being built. A weight is assigned to these features using the `term frequency-inverse document frequency` (TF*IDF) approach .

```
df['text_clean'] = df['Content'].str.lower().str.replace(r'[^w\s]+', ' ').str.replace('\n
    ↪ ', ' ')
vectorizer = TfidfVectorizer(stop_words='english', analyzer='word', ngram_range=(1,3))
```

8.7.1.4 Train

In this example, you skip splitting the dataset into the training and test sets since the goal is to build a toy model. You assign 0 for the `rec.sport.baseball` label and 1 for the `sci.space` label:

```
classifier = DecisionTreeClassifier()
feature = vectorizer.fit_transform(df['text_clean'])
model = classifier.fit(feature, df['Annotations'])
```

8.7.1.5 Predict

Use the following to predict the category for a given text data using the trained binary classifier:

```
classifier.predict(vectorizer.transform(["reggie jackson played right field"]))
```

8.7.2 Image Classification

This example demonstrates how to read image files and labels, normalize the size of the image, train a SVC model, and make predictions. The SVC model is used to try and determine what class a model belongs to.

To start, import the required libraries:

```
import ads
import matplotlib.pyplot as plt
import oci
import os
import pandas as pd

from ads.data_labeling import LabeledDatasetReader
from PIL import Image
from sklearn import svm, metrics
from sklearn.model_selection import train_test_split
```

8.7.2.1 Data Source

The data for this example was taken from a set of x-rays that were previously labeled in the Data Labeling service whether they have pneumonia or not. The metadata was exported and saved in a publicly accessible Object Storage bucket. The following commands define the parameters needed to access the metadata JSONL file:

```
metadata_path = f"oci://hosted-ds-datasets@bigdatadatasciencelarge/DLS/image_single_
↪label_xray/metadata.jsonl"
```

8.7.2.2 Load

This example loads and materializes the data in the dataframe. That is the dataframe to contain a copy of the image file. You do this with the `.ads.read_labeled_data()` method:

```
df = pd.DataFrame.ads.read_labeled_data(path=metadata_path,
                                       materialize=True)
```

8.7.2.3 Visualize

The next example extracts images from the dataframe, and plots them along with their labels:

```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, df.Content, df.Annotations):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title(f'Training: {label}')
```

8.7.2.4 Preprocess

The image files are mixture of RGB and grayscale. Convert all the images to single channel grayscale so that the input to the SVC model is consistent:

```
df.Content = df.Content.apply(lambda x: x.convert("L"))
```

The images are different sizes and you can normalize the size with:

```
basewidth, hsize = min(df.Content.apply(lambda x: x.size))
df.Content = df.Content.apply(lambda x: x.resize((basewidth, hsize), Image.NEAREST))
```

Convert the image to a numpy array as that is what the SVC is expecting. Each pixel in the image is now a dimension in hyperspace.

```
from numpy import asarray
import numpy as np

data = np.stack([np.array(image).reshape(-1) for image in df.Content], axis=0)
labels = df.Annotations
```

The model needs to be trained on one set of data, and then its performance would be assessed on a set of data that it has not seen before. Therefore, this splits the data into a training and testing sets:

```
X_train, X_test, y_train, y_test = train_test_split(
    data, labels, test_size=0.1, shuffle=True)
```

8.7.2.5 Train

The following obtains an SVC classifier object, and trains it on the training set:

```
clf = svm.SVC(gamma=0.001)
clf.fit(X_train, y_train)
```

8.7.2.6 Predict

With the trained SVC model, you can now make predictions using the testing dataset:

```
predicted = clf.predict(X_test)
predicted
```

8.7.3 Multinomial Text Classification

Building a multinomial text classifier is similar to creating a binary text classifier except that you make a classifier for each class. You use a one-vs-the-rest (OvR) multinomial strategy. That is, you create one classifier for each class where one class is the class you are trying to predict, and the other class is all the other classes. You treat the other classes as if they were one class. The classifier predicts whether the observation is in the class or not. If there are m classes, then there will be m classifiers. Classification is based on which classifier has the more confidence that an observation is in the class.

Start by loading in the required libraries:

```
import ads
import nltk
import oci
import os
import pandas as pd

from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.svm import LinearSVC
```

8.7.3.1 Dataset

A subset of the [Reuters Corpus](#) dataset is used in this example. You use scikit-learn and nltk packages to build a multinomial classifier. The Reuters data is a benchmark dataset for document classification. More precisely, it is a data set where the target variable is multinomial. It has 90 categories, 7,769 training documents, and 3,019 testing documents.

The data was previously labeled in the Data Labeling service. The metadata was exported and was saved in a publicly accessible Object Storage bucket. The metadata JSONL file is used to import the data and labels.

8.7.3.2 Load

This example loads a dataset with a target variable that is multinomial. It returns the text and the class annotation in a dataframe:

```
df = pd.DataFrame.ads.read_labeled_data(  
    "oci://hosted-ds-datasets@bigdatadatasciencelarge/DLS/text_multi_label_nltk_reuters/  
↪metadata.jsonl",  
    materialize=True  
)
```

8.7.3.3 Preprocess

You can use the `MultiLabelBinarizer()` method to convert the labels into the scikit-learn classification format during the dataset preprocessing. This [transformer converts](#) a list of sets or tuples into the supported multilabel format, a binary matrix of `samples*classes`.

The next step is to vectorize the input text to feed it into a supervised machine learning system. In this example, TF*IDF vectorization is used.

For performance reasons, the `TfidfVectorizer` is limited to 10,000 words.

```
nlTK.download('stopwords')  
  
stop_words = stopwords.words("english") ## See scikit-learn documentation for what these_  
↪words are  
vectorizer = TfidfVectorizer(stop_words=stop_words, max_features = 10000)  
mlb = MultiLabelBinarizer()  
  
X_train = vectorizer.fit_transform(df["Content"]) ## Vectorize the inputs with tf-idf  
y_train = mlb.fit_transform(df["Annotations"]) ## Vectorize the labels
```

8.7.3.4 Train

You train a Linear Support Vector, `LinearSVC`, classifier using the text data to generate features and annotations to represent the response variable.

The data from the [study class](#) is treated as positive, and the data from all the other classes is treated as negative.

This example uses the scalable Linear Support Vector Machine, `LinearSVC`, for classification. It's quick to train and empirically adequate on NLP problems:

```
clf = OneVsRestClassifier(LinearSVC(class_weight = "balanced"), n_jobs = -1)  
clf.fit(X_train, y_train)
```

8.7.3.5 Predict

The next example applies cross-validation to estimate the prediction error. The K fold cross-validation works by partitioning a dataset into K splits. For the k^{th} part, it fits the model to the other $K-1$ splits of the data and calculates the prediction error. It uses the k^{th} part to do this prediction. For more details about this process, see [here](#) and specifically this [image](#).

By performing cross-validation, there are five separate models trained on different train and test splits to get an estimate of the error that is expected when the model is generalized to an independent dataset. This example uses the `cross_val_score` method to estimate the mean and standard deviation of errors:

```
cross_val_score(clf, X_train, y_train, cv=5)
```

8.7.4 Named Entity Recognition

This example shows you how to use a labeled dataset to create a named entity recognition model. The dataset is labeled using the Oracle Cloud Infrastructure (OCI) Data Labeling Service (DLS).

To start, load the required libraries

```
import ads
import os
import pandas as pd
import spacy

from spacy.tokens import DocBin
from tqdm import tqdm
```

8.7.4.1 Dataset

The [Reuters Corpus](#) is a benchmark dataset that is used in the evaluation of document classification models. It is based on Reuters' financial newswire service articles from 1987. It contains the title and text of the article in addition to a list of people, places and organizations that are referenced in the article. It is this information that is used to label the dataset. A subset of the news articles were labeled using the DLS.

8.7.4.2 Load

This labeled dataset has been exported from the DLS and the metadata has been stored in a publically accessible Object Storage bucket. The `.read_labeled_data()` method is used to load the data. The `materialize` parameter causes the original data to be also be returned with the dataframe.

```
path = 'oci://hosted-ds-datasets@bigdatadatasciencelarge/DLS/text_entity_extraction_nltk_
↳ reuters/metadata.jsonl'
df = pd.DataFrame(ads.read_labeled_data(
    path,
    materialize=True
))
```

8.7.4.3 Preprocess

Covert the annotations data to the [SpaCy format](#) This will give you the start and end position of each entity and then the type of entity, such as person, place, organization.

```
df.Annotations = df.Annotations.apply(lambda items: [x.to_spacy() for x in items])
```

The resulting dataframe will look like the following:

	Path	Content	Annotations
0	oci://hosted-ds-datasets@bigdatadatasciencelar...	(CORRECTED) - MOBIL <MOB> TO UPGRADE REFIN...	[(56, 66, company), (149, 157, city), (161, 16...
1	oci://hosted-ds-datasets@bigdatadatasciencelar...	COFFEE, SUGAR AND COCOA EXCHANGE NAMES CHAIRMA...	[(54, 62, city), (99, 103, company), (140, 146...
2	oci://hosted-ds-datasets@bigdatadatasciencelar...	N.Z. TRADING BANK DEPOSIT GROWTH RISES SLIGHTL...	[(50, 61, country), (189, 201, company)]
3	oci://hosted-ds-datasets@bigdatadatasciencelar...	CANADA OIL EXPORTS RISE 20 PCT IN 1986\nCana...	[(0, 6, country), (41, 49, country), (210, 216...
4	oci://hosted-ds-datasets@bigdatadatasciencelar...	U.K. GROWING IMPATIENT WITH JAPAN - THATCHER\n...	[(62, 79, person), (128, 133, country), (509, ...

In this example, you will not be evaluating the performance of the model. Therefore, the data will not be split into train and test sets. Instead, you use all the data as training data. The following code snippet will create a list of tuples that contain the original article text and the annotation data.

```
train_data = []
for i, row in df.iterrows():
    train_data.append((row['Content'], {'entities': row['Annotations']}))
```

The training data will look similar to the following:

```
[("CORRECTED) - MOBIL &lt;MOB> TO UPGRADE REFINERY UNIT
Mobil Corp said it will spend over 30
mln dlrs to upgrade a gasoline-producing unit at its Beaumont,
...
(Correcting unit's output to barrels/day from barrels/year)",
 {'entities': [(56, 66, 'company'), (149, 157, 'city'), (161, 166, 'city')]}),
('COFFEE, SUGAR AND COCOA EXCHANGE NAMES CHAIRMAN
The New York Coffee, Sugar and Cocoa
...
of Demico Futures, was elected treasurer.',
 {'entities': [(54, 62, 'city'),
 (99, 103, 'company'),
 (140, 146, 'person'),
 (243, 254, 'person'),
 ...
 (718, 732, 'person')]}),
...
]
```

The DocBin format will be used as it provides faster serialization and efficient storage. The following code snippet does the conversion and writes the resulting DocBin object to a file.


```

nlp = spacy.blank("en") # load a new spacy model
db = DocBin() # create a DocBin object
i=0
for text, annot in tqdm(train_data): # data in previous format
    doc = nlp.make_doc(text) # create doc object from text
    ents = []
    for start, end, label in annot["entities"]: # add character indexes
        span = doc.char_span(start, end, label=label, alignment_mode="contract")

        if span is not None:
            ents.append(span)
    doc.ents = ents # label the text with the ents
    db.add(doc)

db.to_disk(os.path.join(os.path.expanduser("~"), "train.spacy") # save the docbin object

```

8.7.4.4 Train

The model will be trained using spaCy. Since this is done through the command line a configuration file is needed. In spaCy, this is a two-step process. You will create a `base_config.cfg` file that will contain the non-default settings for the model. Then the `init fill-config` argument on the spaCy module will be used to auto-fill a partial config. `cfg` file with the default values for the parameters that are not given in the `base_config.cfg` file. The `config.cfg` file contains all the settings and hyperparameters that will be needed to train the model. See the [spaCy training documentation](#) for more details.

The following code snippet will write the `base_config.cfg` configuration file and contains all the non-default parameter values.

```

config = """
[paths]
train = null
dev = null

[system]
gpu_allocator = null

[nlp]
lang = "en"
pipeline = ["tok2vec", "ner"]
batch_size = 1000

[components]

[components.tok2vec]
factory = "tok2vec"

[components.tok2vec.model]
@architectures = "spacy.Tok2Vec.v2"

[components.tok2vec.model.embed]
@architectures = "spacy.MultiHashEmbed.v2"
width = ${components.tok2vec.model.encode.width}

```

(continues on next page)

(continued from previous page)

```
attrs = ["ORTH", "SHAPE"]
rows = [5000, 2500]
include_static_vectors = false

[components.tok2vec.model.encode]
@architectures = "spacy.MaxoutWindowEncoder.v2"
width = 96
depth = 4
window_size = 1
maxout_pieces = 3

[components.ner]
factory = "ner"

[components.ner.model]
@architectures = "spacy.TransitionBasedParser.v2"
state_type = "ner"
extra_state_tokens = false
hidden_width = 64
maxout_pieces = 2
use_upper = true
n0 = null

[components.ner.model.tok2vec]
@architectures = "spacy.Tok2VecListener.v1"
width = ${components.tok2vec.model.encode.width}

[corpora]

[corpora.train]
@readers = "spacy.Corpus.v1"
path = ${paths.train}
max_length = 0

[corpora.dev]
@readers = "spacy.Corpus.v1"
path = ${paths.dev}
max_length = 0

[training]
dev_corpus = "corpora.dev"
train_corpus = "corpora.train"

[training.optimizer]
@optimizers = "Adam.v1"

[training.batcher]
@batchers = "spacy.batch_by_words.v1"
discard_oversize = false
tolerance = 0.2

[training.batcher.size]
```

(continues on next page)

(continued from previous page)

```
@schedules = "compounding.v1"
start = 100
stop = 1000
compound = 1.001

[initialize]
vectors = ${paths.vectors}
"""

with open(os.path.join(os.path.expanduser("~"), "base_config.cfg"), 'w') as f:
    f.write(config)
```

The following code snippet calls a new Python interpreter that runs the spaCy module. It loads the `base_config.cfg` file and writes out the configuration file `config.cfg` that has all of the training parameters that will be used. It contains the default values plus the ones that were specified in the `base_config.cfg` file.

```
!$CONDA_PREFIX/bin/python -m spacy init fill-config ~/base_config.cfg ~/config.cfg
```

To train the model, you will call a new Python interpreter to run the spaCy module using the `train` command-line argument and other arguments that point to the training files that you have created.

```
!$CONDA_PREFIX/bin/python -m spacy train ~/config.cfg --output ~/output --paths.train ~/
↪train.spacy --paths.dev ~/train.spacy
```

8.7.4.5 Predict

The spaCy training procedure creates a number of models. The best model is stored in `model-best` under the output directory that was specified. The following code snippet loads that model and creates a sample document. The model is run and the output has the new document plus and entities that were detected are highlighted.

```
nlp = spacy.load(os.path.join(os.path.expanduser("~"), "output", "model-best")) #load the
↪best model
doc = nlp("The Japanese minister for post and telecommunications was reported as saying
↪that he opposed Cable and Wireless having a managerial role in the new company.") #
↪input sample text

spacy.displacy.render(doc, style="ent", jupyter=True) # display in Jupyter
```

The Japanese minister for post and telecommunications was reported as saying that he opposed **Cable and Wireless company** having a managerial role in the new company.

TRANSFORM DATA

When datasets are loaded with DatasetFactory, they can be transformed and manipulated easily with the built-in functions. Underlying, an ADSDataset object is a Pandas dataframe. Any operation that can be performed to a [Pandas dataframe](#) can also be applied to an ADS Dataset.

9.1 Loading the Dataset

You can load a pandas dataframe into an ADSDataset by calling.

```
from ads.dataset.factory import DatasetFactory  
  
ds = DatasetFactory.from_dataframe(df)
```

9.2 Automated Transformations

ADS has built in automatic transform tools for datasets. When the `get_recommendations()` tool is applied to an ADSDataset object, it shows the user detected issues with the data and recommends changes to apply to the dataset. You can accept the changes as easy as clicking a button in the drop down menu. After all the changes are applied, the transformed dataset can be retrieved by calling `get_transformed_dataset()`.

```
wine_ds.get_recommendations()
```

Alternatively, you can use `auto_transform()` to apply all the recommended transformations at once. `auto_transform()` returns a transformed dataset with several optimizations applied automatically. The optimizations include:

- Dropping constant and primary key columns, which has no predictive quality.
- Imputation to fill in missing values in noisy data.
- Dropping strongly co-correlated columns that tend to produce less generalizable models.
- Balancing a dataset using up or down sampling.

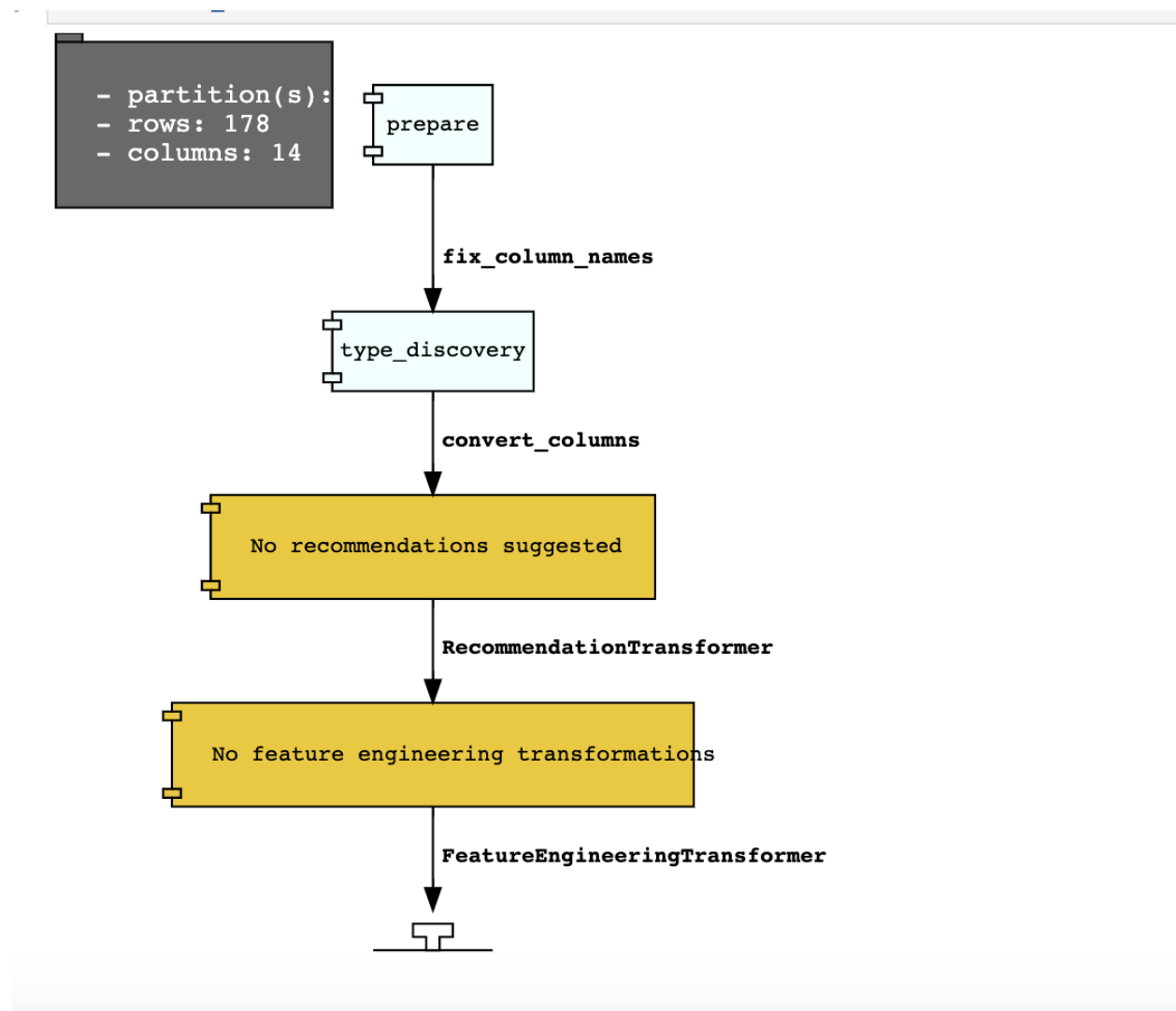
One optional argument to `auto_transform()` is `fix_imbalance`, which is set to `True` by default. When `True`, `auto_transform()` corrects any imbalance between the classes. ADS downsamples the dominant class first unless there are too few data points. In that case, ADS upsamples the minority class.

```
ds = wine_ds.auto_transform()
```

You can visualize the transformation that has been performed on a dataset by calling `visualize_transforms()`.

Note: `visualize_transforms()` is only applied to the automated transformations and does not capture any custom transformations that you may have applied to the dataset.

```
ds.visualize_transforms()
```



9.3 Row Operations

The operations that can be applied to a Pandas dataframe can be applied to an `ADSDataset` object.

Examples of some of the most common row operations you can apply on an `ADSDataset` object follow.

9.3.1 Delete Rows

Rows within a dataset can be filtered out by row numbers. The index of the new dataset can be reset accordingly.

```
#Filter out rows by row number and reset index of new data
ds_subset = ds.loc[10:100]
ds_subset = ds_subset.reset_index()
```

Do not try to insert index into dataset columns.

9.3.2 Reset Index

Reset the index to the default index. When you reset index, the old index is added as a column `index` and a new sequential index is used. You can use the `drop` parameter to avoid the old index being added as a column:

```
ds_subset = ds.loc[10:100]
ds_subset = ds_subset.reset_index(drop=True)
ds_subset.head()
```

The index restarts at zero for each partition. This is due to the inability to statically know the full length of the index.

9.3.3 Append Rows

New rows can be added to an existing dataset:

```
#Create new row to be added
row_to_add = ds.loc[0]
row_to_add['target'] = 'class_0'

#Add in new row to existing dataset
new_addition_ds = ds.merge(row_to_add, how = 'outer')
```

Alternatively, you can use the `append()` method of a Pandas dataframe to achieve a similar result:

```
ds2 = wine_ds.df.append(ds)
```

The `ds2` is created as a Pandas DataFrame object.

9.3.4 Row Filtering

Columns can be filtered out by the values:

```
ds_filtered = ds[(ds['alcohol'] > 13.0) & (ds['malic_acid'] < 2.5)]
ds_filtered.head()
```

9.3.5 Removing Duplicated Rows

Duplicate rows can be removed using the `drop_duplicates` function:

```
ds_without_dup = ds.drop_duplicates()
```

9.4 Column Operations

The column operations that can be applied to a Pandas dataframe can be applied to an ADS dataset as in the following examples.

9.4.1 Delete a Column

To delete specific columns from the dataset, the `drop_columns` function can be used along with names of the columns to be deleted from the dataset. The `ravel` Pandas command returns the flattened underlying data as an ndarray. The `name_of_df.columns[:].ravel()` command returns the name of all the columns in a dataframe as an array.

```
ds_subset_columns = ds.drop_columns(['alcohol', 'malic_acid'])
ds_subset_columns.columns[:].ravel()
```

```
array(['ash', 'alkalinity_of_ash', 'magnesium', 'total_phenols',
       'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
       'color_intensity', 'hue', 'od280/od315_of_diluted_wines',
       'proline', 'target'], dtype=object)
```

9.4.2 Rename a Column

Columns can be renamed with the `rename_columns()` method:

```
ds_columns_rename = ds.rename_columns({'alcohol': 'alcohol_amount',
                                       'malic_acid': 'malic_acid_amount'})
ds_columns_rename.columns[:].ravel()
```

```
array(['alcohol_amount', 'malic_acid_amount', 'ash', 'alkalinity_of_ash',
       'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
       'proanthocyanins', 'color_intensity', 'hue',
       'od280/od315_of_diluted_wines', 'proline', 'target'], dtype=object)
```

9.4.3 Counts of Unique Values

The count per unique value can be obtained with the `value_counts()` method:

```
ds['target'].value_counts()
```

```
class_1    71
class_0    59
class_2    48
Name: target, dtype: int64
```


9.4.4 Normalize a Column

You can apply a variety of normalization techniques to numerical columns (both continuous and discrete). You can leverage the built in `max()` and `min()` methods to perform a minmax normalization:

```
max_alcohol = wine_ds['alcohol'].max()
min_alcohol = wine_ds['alcohol'].min()
alcohol_range = max_alcohol - min_alcohol
wine_ds.df['norm_alcohol'] = (wine_ds['alcohol'] / alcohol_range)
```

9.4.5 Combine Columns

This example creates a new column by performing operations to combine two or more columns together:

```
new_feature_col = ((0.4)*wine_ds['total_phenols'] + (0.6)*wine_ds['flavanoids'])
ds_new_feature = wine_ds.assign_column('new_feature', new_feature_col)
ds_new_feature.head()
```

Alternatively, you can create a new column directly in the Pandas dataframe attribute:

```
new_feature_col = ((0.4)*wine_ds['total_phenols'] + (0.6)*wine_ds['flavanoids'])
wine_ds.df['new_feature'] = new_feature_col
wine_ds.head()
```

To add new column, use a new name for it. You can add anew column and change it by combining with existing column:

```
noise = np.random.normal(0,.1,wine_ds.shape[0])
ds_noise = wine_ds.assign_column('noise', noise)

ds_ash = ds_noise.assign_column('noise', ds_noise['noise'] + ds_noise['ash'])
ds_ash = ds_ash.rename(columns={'noise':'ash_with_noise'})
ds_ash.head()
```

The resulting column is renamed with dict-like mapper.

9.4.6 Apply a Function to a Column

You can apply functions to update column values in existing column. This example updates the column in place using lambda expression:

```
wine_ds.assign_column('proline', lambda x: x is None or x > 1000)
wine_ds.head()
```

9.4.7 Change Data Type

You can change the data type columns with the `astype()` method. ADS uses the Pandas method, `astype()`, on dataframe objects. For specifics, see [astype for a Pandas Dataframe, using `numpy.dtype`, or Pandas `dtypes`](#).

When you change the type of a column, ADS updates its semantic type to categorical, continuous, datetime, or ordinal. For example, if you update a column type to integer, its semantic type updates to ordinal. For data type details, see [ref:loading-data-specify-dtype](#).

This example converts a dataframe column from float, to the low-level integer type and ADS updates its semantic type to ordinal:

```
wine_ds = wine_ds.astype(types={'proline': 'int64'})
print(wine_ds.feature_types['proline']['low_level_type'])
print(wine_ds.feature_types['proline']['type'])

# Note: When you cast a float column to integer, you lose precision.
wine_ds['proline'].head()
```

To convert a column of type float to categorical, you convert it to integer first. This example converts a column data type from float to integer, then to categorical, and then the number of categories in the column is reduced:

```
# create a new dataset with a renamed column for binned data and update the values
ds = wine_ds.rename_columns({'color_intensity': 'color_intensity_bin'})
ds = ds.assign_column('color_intensity_bin', lambda x: x/3)

# convert the column from float to categorical:
ds = ds.astype(types={'color_intensity_bin': 'int64'})
ds = ds.astype(types={'color_intensity_bin': 'categorical'})
```

You can use `feature_types` to see if the semantic data type of the converted column is categorical:

```
wine_ds.feature_types['color_intensity_bin']['type']
```

```
'categorical'
```

The low-level type of the converted column is category:

```
ds['color_intensity_bin'].head()
```

```
0    1
1    1
2    1
3    2
4    1
Name: color_intensity_bin, dtype: category
Categories (5, int64): [0, 1, 2, 3, 4]
```

9.5 Dataset Manipulation

ADS has built in functions that support categorical encoding, null values and imputation.

9.5.1 Categorical Encoding

ADS has a built in categorical encoder that can be accessed by calling from `ads.dataset.label_encoder` import `DataFrameLabelEncoder`. This example encodes the three classes of wine that make up the dataset:

```
from ads.dataset.label_encoder import DataFrameLabelEncoder
ds_encoded = DataFrameLabelEncoder().fit_transform(ds.to_pandas())
ds_encoded['target'].value_counts()
```

```
1    71
0    59
2    48
```

9.5.2 One-Hot Encoding

One-hot encoding transforms one categorical column with n categories into n or $n-1$ columns with indicator variables. You can prepare one of the columns to be categorical with categories low, medium, and high:

```
def convert_to_level(value):
    if value < 12:
        return 'low'
    elif value > 13:
        return 'high'
    else:
        return 'medium'

ds = wine_ds
ds = ds.assign_column('alcohol', convert_to_level)
```

You can use the Pandas method `get_dummies()` to perform one-hot encoding on a column. Use the `prefix` parameter to assign a prefix to the new columns that contain the indicator variables. This example creates n columns with one-hot encoding:

```
data = ds.to_pandas()['alcohol'] # data of which to get dummy indicators
onehot = pd.get_dummies(data, prefix='alcohol')
```

To create $n-1$ columns, use `drop_first=True` when converting the categorical column. You can add a one-hot column to the initial dataset with the `merge()` method:

```
data = ds.to_pandas()['alcohol'] # data of which to get dummy indicators
onehot = pd.get_dummies(data, prefix='alcohol', drop_first=False)
ds_onehot = ds.merge(onehot)
```

Encoding for all categorical columns can be accomplished with the `fit_transform()` method:

```
from ads.dataset.label_encoder import DataFrameLabelEncoder
```

(continues on next page)

(continued from previous page)

```
ds_encoded = DataFrameLabelEncoder().fit_transform(ds_onehot.to_pandas())
ds_encoded['alcohol'].value_counts()
```

```
0    92
2    67
1    19
```

To drop the initial categorical column that you transformed into one-hot, use one of these examples:

```
ds_onehot = ds_onehot.drop_columns('alcohol') # before ``fit_transform()`` method
# or
ds_encoded = ds_encoded.drop(columns='alcohol') # after ``fit_transform()`` method
```

9.5.3 Extract Null Values

To detect all nulls in a dataset, use the `isnull` function to return a boolean dataset matching the dimension of our input:

```
ds_null = ds.isnull()
np.any(ds_null)
```

```
alcohol           False
malic_acid        False
ash              False
alcalinity_of_ash False
magnesium        False
total_phenols     False
flavanoids       False
nonflavanoid_phenols False
proanthocyanins  False
color_intensity  False
hue              False
od280/od315_of_diluted_wines False
proline          False
target           False
```

9.5.4 Imputation

The `fillna` function is used to replace null values with specific values. Generate a null value by replacing the entry below a certain value with null, and then imputing it with a value:

```
ds_with_null = ds.assign_column("malic_acid", lambda x: None if x < 2 else x)
ds_with_null['malic_acid'].head()
```

```
0    NaN
1    NaN
2    2.36
3    NaN
```

(continues on next page)

(continued from previous page)

```
4    2.59
Name: malic_acid, dtype: float64
```

```
ds_impute = ds_with_null.fillna(method='bfill')
ds_impute['malic_acid'].head()
```

```
0    2.36
1    2.36
2    2.36
3    2.59
4    2.59
Name: malic_acid, dtype: float64
```

9.5.5 Combine Datasets

ADS datasets can be merged and combined together to form a new dataset.

9.5.5.1 Join Datasets

You can merge two datasets together with a database-styled join on columns or indexes by specifying the type of join `left`, `right`, `outer`, or `inner`. These type are defined by:

- `left`: Use only keys from the left dataset, similar to SQL left outer join.
- `right`: Use only keys from the right dataset, similar to SQL right outer join.
- `inner`: Intersection of keys from both datasets, similar to SQL inner join.
- `outer`: Union of keys from both datasets, similar to SQL outer join.

This is an example of performing an outer join on two datasets. The datasets are subsets of the wine dataset, and each dataset contains only one class of wine.

```
ds_class1 = ds[ds['target']=='class_1']
ds_class2 = ds[ds['target']=='class_2']
ds_merged_outer = ds_class1.merge(ds_class2, how='outer')
ds_merged_outer['target'].value_counts()
```

```
class_1    71
class_2    48
class_0     0
Name: target, dtype: int64
```

9.5.5.2 Concatenate Datasets

Two datasets can be concatenated along a particular axis (vertical or horizontal) with the option of performing set logic (union or intersection) of the indexes on the other axes. You can stack two datasets vertically with:

```
ds_concat = pd.concat([ds_class1, ds_class2], axis = 0)
ds_concat['target'].value_counts()
```

```
class_1    71
class_2    48
class_0     0
Name: target, dtype: int64
```

9.6 Train/Test Datasets

After all data transformations are complete, you can split the data into a train and test or train, test, and validation set. To split data into a train and test set with a train size of 80% and test size of 20%:

```
from ads.dataset.dataset_browser import DatasetBrowser
sklearn = DatasetBrowser.sklearn()
wine_ds = sklearn.open('wine')
ds = wine_ds.auto_transform()
train, test = ds.train_test_split(test_size=0.2)
```

For a train, test, and validation set, the defaults are set to 80% of the data for training, 10% for testing, and 10% for validation. This example sets split to 70%, 15%, and 15%:

```
data_split = wine_ds.train_validation_test_split(
    test_size=0.15,
    validation_size=0.15
)
train, validation, test = data_split
print(data_split)  # print out shape of train, validation, test sets in split
```

The resulting three data subsets each have separate data (X) and labels (y).

```
print(train.X)  # print out all features in train dataset
print(train.y)  # print out labels in train dataset
```

You can split the dataset right after the `DatasetFactory.open()` statement:

```
ds = DatasetFactory.open("path/data.csv").set_target('target')
train, test = ds.train_test_split(test_size=0.25)
```

9.7 Text Data

9.7.1 TextStrings

9.7.1.1 Overview

Text analytics uses a set of powerful tools to understand the content of unstructured data, such as text. It's becoming an increasingly more important tool in feature engineering as product reviews, media content, research papers, and more are being mined for their content. In many data science areas, such as marketing analytics, the use of unstructured text is becoming as popular as structured data. This is largely due to the relatively low cost of collection of the data. However, the downside is the complexity of working with the data. To work with unstructured that you need to clean, summarize, and create features from it before you create a model. The `ADSString` class provides tools that allow you to quickly do this work. More importantly, you can expand the tool to meet your specific needs.

Data scientists need to be able to quickly and easily manipulate strings. ADS SDK provides an enhanced string class, called `ADSString`. It adds functionality like regular expression (RegEx) matching and natural language processing (NLP) parsing. The class can be expanded by registering custom plugins so that you can process a string in a way that it fits your specific needs. For example, you can register the [OCI Language service](#) plugin to bind functionalities from the [OCI Language service](#) to `ADSString`.

9.7.1.2 Quick Start

9.7.1.2.1 NLP Parse

The following example parses a text corpus using the `NLTK` and `spaCy` engines.

```
from ads.feature_engineering.adsstring.string import ADSString

s = ADSString("""
Lawrence Joseph Ellison (born August 17, 1944) is an American business magnate,
investor, and philanthropist who is a co-founder, the executive chairman and
chief technology officer (CTO) of Oracle Corporation. As of October 2019, he was
listed by Forbes magazine as the fourth-wealthiest person in the United States
and as the sixth-wealthiest in the world, with a fortune of $69.1 billion,
increased from $54.5 billion in 2018.[4] He is also the owner of the 41st
largest island in the United States, Lanai in the Hawaiian Islands with a
population of just over 3000.
""").strip())

# NLTK
ADSString.nlp_backend("nltk")
noun = s.noun
adj = s.adjective
pos = s.pos # Parts of Speech

# spaCy
ADSString.nlp_backend("spacy")
noun = s.noun
adj = s.adjective
pos = s.pos # Parts of Speech
```

9.7.1.2.2 Plugin

Custom Plugin

This example demonstrates how to create a custom plugin that will take a string, detect the credit card numbers, and return a list of the last four digits of the credit card number.

```
from ads.feature_engineering.adsstring.string import ADSSString

class CreditCardLast4:
    @property
    def credit_card_last_4(self):
        return [x[len(x)-4:len(x)] for x in ADSSString(self.string).credit_card]

ADSSString.plugin_register(CreditCardLast4)

creditcard_numbers = "I purchased the gift on this card 4532640527811543 and the dinner_
↳ on 340984902710890"
s = ADSSString(creditcard_numbers)
s.credit_card_last_4
```

OCI Language Services Plugin

This example uses the [OCI Language service](#) to perform an aspect-based sentiment analysis, language detection, key phrase extraction, and a named entity recognition.

```
from ads.feature_engineering.adsstring.oci_language import OCILanguage
from ads.feature_engineering.adsstring.string import ADSSString

ADSSString.plugin_register(OCILanguage)

s = ADSSString("""
    Lawrence Joseph Ellison (born August 17, 1944) is an American business magnate,
    investor, and philanthropist who is a co-founder, the executive chairman and
    chief technology officer (CTO) of Oracle Corporation. As of October 2019, he was
    listed by Forbes magazine as the fourth-wealthiest person in the United States
    and as the sixth-wealthiest in the world, with a fortune of $69.1 billion,
    increased from $54.5 billion in 2018.[4] He is also the owner of the 41st
    largest island in the United States, Lanai in the Hawaiian Islands with a
    population of just over 3000.
    """).strip()

# Aspect-Based Sentiment Analysis
df_sentiment = s.absa

# Key Phrase Extraction
key_phrase = s.key_phrase

# Language Detection
language = s.language_dominant
```

(continues on next page)

(continued from previous page)

```
# Named Entity Recognition
named_entity = s.ner

# Text Classification
classification = s.text_classification
```

9.7.1.2.3 RegEx Match

In this example, the dates and prices are extracted from the text using regular expression matching.

```
from ads.feature_engineering.adsstring.string import ADSString

s = ADSString("""
    Lawrence Joseph Ellison (born August 17, 1944) is an American business magnate,
    investor, and philanthropist who is a co-founder, the executive chairman and
    chief technology officer (CTO) of Oracle Corporation. As of October 2019, he was
    listed by Forbes magazine as the fourth-wealthiest person in the United States
    and as the sixth-wealthiest in the world, with a fortune of $69.1 billion,
    increased from $54.5 billion in 2018.[4] He is also the owner of the 41st
    largest island in the United States, Lanai in the Hawaiian Islands with a
    population of just over 3000.
    """).strip()

dates = s.date
prices = s.price
```

9.7.1.3 NLP Parse

ADSString also supports NLP parsing and is backed by [Natural Language Toolkit \(NLTK\)](#) or [spaCy](#). Unless otherwise specified, NLTK is used by default. You can extract properties, such as nouns, adjectives, word counts, parts of speech tags, and so on from text with NLP.

The ADSString class can have one backend enabled at a time. What properties are available depends on the backend, as do the results of calling the property. The following examples provide an overview of the available parsers, and how to use them. Generally, the parser supports the adjective, adverb, bigram, noun, pos, sentence, trigram, verb, word, and word_count base properties. Parsers can support additional parsers.

9.7.1.3.1 NLTK

The Natural Language Toolkit ([NLTK](#)) is a powerful platform for processing human language data. It supports all the base properties and in addition `stem` and `token`. The `stem` property returns a list of all the stemmed tokens. It reduces a token to its word stem that affixes to suffixes and prefixes, or to the roots of words that is the lemma. The `token` property is similar to the `word` property, except it returns non-alphanumeric tokens and doesn't force tokens to be lowercase.

The following example use a sample of text about Larry Ellison to demonstrate the use of the NLTK properties.

```
test_text = """
    Lawrence Joseph Ellison (born August 17, 1944) is an American business
    ↪magnate,
```

(continues on next page)

(continued from previous page)

```

investor, and philanthropist who is a co-founder, the executive chairman and
chief technology officer (CTO) of Oracle Corporation. As of October 2019, he
↳was
listed by Forbes magazine as the fourth-wealthiest person in the United
↳States
and as the sixth-wealthiest in the world, with a fortune of $69.1 billion,
increased from $54.5 billion in 2018.[4] He is also the owner of the 41st
largest island in the United States, Lanai in the Hawaiian Islands with a
population of just over 3000.
"".strip()
ADSSString.nlp_backend("nltk")
s = ADSSString(test_text)

```

```
s.noun[1:5]
```

```
['Joseph', 'Ellison', 'August', 'business']
```

```
s.adjective
```

```
['American', 'chief', 'fourth-wealthiest', 'largest', 'Hawaiian']
```

```
s.word[1:5]
```

```
['joseph', 'ellison', 'born', 'august']
```

By taking the difference between token and word, the token set contains non-alphanumeric tokens, and also the upper-case version of words.

```
list(set(s.token) - set(s.word))[1:5]
```

```
['Oracle', '1944', '41st', 'fourth-wealthiest']
```

The `stem` property takes the list of words and stems them. It produces morphological variations of a word's root form. The following example stems some words, and shows some of the stemmed words that were changed.

```
list(set(s.stem) - set(s.word))[1:5]
```

```
['fortun', 'technolog', 'increas', 'popul']
```

Part of Speech Tags

Part of speech (POS) is a category in which a word is assigned based on its syntactic function. POS depends on the language. For English, the most common POS are adjective, adverb, conjunction, determiner, interjection, noun, preposition, pronoun, and verb. However, each POS system has its own set of POS tags that vary based on their respective training set. The NLTK parsers produce the following POS tags:

Table 1: Parts of Speech Tags

CC: coordinating conjunction	CD: cardinal digit
DT: determiner	EX: existential there; like “there is”; “there exists”
FW: foreign word	IN: preposition/subordinating conjunction
JJ: adjective; “big”	JJR: adjective, comparative; “bigger”
JJS: adjective, superlative; “biggest”	LS: list marker 1)
MD: modal could, will	NN: noun, singular; “desk”
NNS: noun plural; “desks”	NNP: proper noun, singular; “Harrison”
NNPS: proper noun, plural; “Americans”	PDT: predeterminer; “all the kids”
POS: possessive ending; “parent’s”	PRP: personal pronoun; I, he, she
PRP\$: possessive pronoun; my, his, hers	RB: adverb; very, silently
RBR: adverb; comparative better	RBS: adverb; superlative best
RP: particle; give up	TO: to go; “to” the store.
UH: interjection; errrrrrrm	VB: verb, base form; take
VBD: verb, past tense; took	VBG: verb, gerund/present participle; taking
VCN: verb, past participle; taken	VBP: verb, singular present; non-3d take
VBZ: verb, 3rd person singular present; takes	WDT: wh-determiner; which
WP: wh-pronoun; who, what	WP\$: possessive wh-pronoun; whose
WRB: wh-adverb; where, when	

```
s.pos[1:5]
```

	Word	Label
1	Joseph	NNP
2	Ellison	NNP
3	((
4	born	VCN

9.7.1.3.2 spaCy

spaCy is an advanced NLP toolkit. It helps you understand what the words mean in context, and who is doing what to whom. It helps you determine what companies and products are mentioned in a document. The spaCy backend is used to parse the adjective, adverb, bigram, noun, pos, sentence, trigram, verb, word, and word_count base properties. It also supports the following additional properties:

- **entity:** All entities in the text.
- **entity_artwork:** The titles of books, songs, and so on.
- **entity_location:** Locations, facilities, and geopolitical entities, such as countries, cities, and states.
- **entity_organization:** Companies, agencies, and institutions.
- **entity_person:** Fictional and real people.
- **entity_product:** Product names and so on.
- **lemmas:** A rule-based estimation of the roots of a word.

- `tokens`: The base tokens of the tokenization process. This is similar to `word`, but it includes non-alphanumeric values and the word case is preserved.

If the `spacy` module is installed, you can change the NLP backend using the `ADSSString.nlp_backend('spacy')` command.

```
ADSSString.nlp_backend("spacy")
s = ADSSString(test_text)
```

```
s.noun[1:5]
```

```
['magnate', 'investor', 'philanthropist', 'co']
```

```
s.adjective
```

```
['American', 'executive', 'chief', 'fourth', 'wealthiest', 'largest']
```

```
s.word[1:5]
```

```
['Joseph', 'Ellison', 'born', 'August']
```

You can identify all the locations that are mentioned in the text.

```
s.entity_location
```

```
['the United States', 'the Hawaiian Islands']
```

Also, the organizations that were mentioned.

```
s.entity_organization
```

```
['CTO', 'Oracle Corporation', 'Forbes', 'Lanai']
```

Part of Speech Tags

The POS tagger in `spacy` uses a smaller number of categories. For example, `spacy` has the `ADJ` POS for all adjectives, while `NLTK` has `JJ` to mean an adjective. `JJR` refers to a comparative adjective, and `JJS` refers to a superlative adjective. For fine grain analysis of different parts of speech, `NLTK` is the preferred backend. However, `spacy`'s reduced category set tends to produce fewer errors, at the cost of not being as specific.

The `spacy` parsers produce the following POS tags:

- `ADJ`: adjective; big, old, green, incomprehensible, first
- `ADP`: adposition; in, to, during
- `ADV`: adverb; very, tomorrow, down, where, there
- `AUX`: auxiliary; is, has (done), will (do), should (do)
- `CONJ`: conjunction; and, or, but
- `CCONJ`: coordinating conjunction; and, or, but
- `DET`: determiner; a, an, the

- INTJ: interjection; psst, ouch, bravo, hello
- NOUN: noun; girl, cat, tree, air, beauty
- NUM: numeral; 1, 2017, one, seventy-seven, IV, MMXIV
- PART: particle; 's, not,
- PRON: pronoun; I, you, he, she, myself, themselves, somebody
- PROP: proper noun; Mary, John, London, NATO, HBO
- PUNCT: punctuation; ., (,), ?
- SCONJ: subordinating conjunction; if, while, that
- SYM: symbol; \$, %, ¢, +, , ×, ÷, =, :),
- VERB: verb; run, runs, running, eat, ate, eating
- X: other; sfpkdpsxmsa
- SPACE: space

```
s.pos[1:5]
```

	Word	Label
1	Joseph	PROP
2	Ellison	PROP
3	(PUNCT
4	born	VERB

9.7.1.4 Plugin

One of the most powerful features of `ADSSString` is that you can expand and customize it. The `.plugin_register()` method allows you to add properties to the `ADSSString` class. These plugins can be provided by third-party providers or developed by you. This section demonstrates how to connect the to the [OCI Language service](#), and how to create a custom plugin.

9.7.1.4.1 Custom Plugin

You can bind additional properties to `ADSSString` using custom plugins. This allows you to create custom text processing extensions. A plugin has access to the `self.string` property in `ADSSString` class. You can define functions that perform a transformation on the text in the object. All functions defined in a plugin are bound to `ADSSString` and accessible across all objects of that class.

Assume that your text is "I purchased the gift on this card 4532640527811543 and the dinner on 340984902710890" and you want to know what credit cards were used. The `.credit_card` property returns the entire credit card number. However, for privacy reasons you don't want the entire credit card number, but the last four digits.

To solve this problem, you can create the class `CreditCardLast4` and use the `self.string` property in `ADSSString` to access the text associated with the object. It then calls the `.credit_card` method to get the credit card numbers. Then it parses this to return the last four characters in each credit card.

The first step is to define the class that you want to bind to `ADSSString`. Use the `@property` decorator and define a property function. This function only takes `self`. The `self.string` is accessible with the text that is defined for a given object. The property returns a list.

```
class CreditCardLast4:
    @property
    def credit_card_last_4(self):
        return [x[len(x)-4:len(x)] for x in ADSSString(self.string).credit_card]
```

After the class is defined, it must be registered with `ADSSString` using the `.register_plugin()` method.

```
ADSSString.plugin_register(CreditCardLast4)
```

Take the text and make it an `ADSSString` object, and call the `.credit_card_last_4` property to obtain the last four digits of the credit cards that were used.

```
creditcard_numbers = "I purchased the gift on this card 4532640527811543 and the dinner_↪on 340984902710890"
s = ADSSString(creditcard_numbers)
s.credit_card_last_4
```

```
['1543', '0890']
```

9.7.1.4.2 OCI Language Services

The `OCI Language service` provides pretrained models that provide sophisticated text analysis at scale.

The Language service contains these pretrained language processing capabilities:

- **Aspect-Based Sentiment Analysis:** Identifies aspects from the given text and classifies each into positive, negative, or neutral polarity.
- **Key Phrase Extraction:** Extracts an important set of phrases from a block of text.
- **Language Detection:** Detects languages based on the given text, and includes a confidence score.
- **Named Entity Recognition:** Identifies common entities, people, places, locations, email, and so on.
- **Text Classification:** Identifies the document category and subcategory that the text belongs to.

Those are accessible in ADS using the `OCILanguage` plugin.

```
ADSSString.plugin_register(OCILanguage)
```

Aspect-Based Sentiment Analysis

Aspect-based sentiment analysis can be used to gauge the mood or the tone of the text.

The aspect-based sentiment analysis (ABSA) supports fine-grained sentiment analysis by extracting the individual aspects in the input document. For example, a restaurant review “The driver was really friendly, but the taxi was falling apart.” contains positive sentiment toward the taxi driver aspect. Also, it has a strong negative sentiment toward the service mechanical aspect of the taxi. Classifying the overall sentiment as negative would neglect the fact that the taxi driver was nice.

ABSA classifies each of the aspects into one of the three polarity classes, positive, negative, mixed, and neutral. With the predicted sentiment for each aspect. It also provides a confidence score for each of the classes and their corresponding

offsets in the input. The range of the confidence score for each class is between 0 and 1, and the cumulative scores of all the three classes sum to 1.

In the next example, the sample sentence is analyzed. The two aspects, taxi cab and driver, have their sentiments determined. It defines the location of the aspect by giving its offset position in the text, and the length of the aspect in characters. It also gives the text that defines the aspect along with the sentiment scores and which sentiment is dominant.

```
t = ADSString("The driver was really friendly, but the taxi was falling apart.")
t.absa
```

	Length	Offset	Sentiment	Text	Negative	Neutral	Positive
0	6	4	Positive	driver	0.0	3.484637e-09	1.000000e+00
1	4	40	Negative	taxi	1.0	0.000000e+00	5.187591e-10

Key Phrase Extraction

Key phrase (KP) extraction is the process of extracting the words with the most relevance, and expressions from the input text. It helps summarize the content and recognizes the main topics. The KP extraction finds insights related to the main points of the text. It understands the unstructured input text, and returns keywords and KPs. The KPs consist of subjects and objects that are being talked about in the document. Any modifiers, like adjectives associated with these subjects and objects, are also included in the output. Confidence scores for each key phrase that signify how confident the algorithm is that the identified phrase is a KP. Confidence scores are a value from 0 to 1.

The following example determines the key phrases and the importance of these phrases in the text (which is the value of `test_text`):

```
Lawrence Joseph Ellison (born August 17, 1944) is an American business magnate, investor, and philanthropist who is a co-founder, the executive chairman and chief technology officer (CTO) of Oracle Corporation. As of October 2019, he was listed by Forbes magazine as the fourth-wealthiest person in the United States and as the sixth-wealthiest in the world, with a fortune of $69.1 billion, increased from $54.5 billion in 2018.[4] He is also the owner of the 41st largest island in the United States, Lanai in the Hawaiian Islands with a population of just over 3000.
```

```
s = ADSString(test_text)
s.key_phrase
```

	Score	Text
0	1.000000	united states
1	0.999811	lawrence joseph ellison
2	0.999811	august 17
3	0.999811	american business magnate
4	0.999811	executive chairman
5	0.999811	chief technology officer
6	0.999811	oracle corporation
7	0.999811	october 2019
8	0.999811	forbes magazine
9	0.999811	fourth-wealthiest person
10	0.999811	fortune of \$69.1 billion
11	0.999811	41st largest island
12	0.999811	hawaiian islands
13	0.999239	philanthropist
14	0.999239	co-founder
15	0.999239	cto
16	0.999239	sixth-wealthiest
17	0.999239	lanai
18	0.999239	population
19	0.997934	investor
20	0.973272	owner

Language Detection

The language detection tool identifies which natural language the input text is in. If the document contains more than one language, the results may not be what you expect. Language detection can help make customer support interactions more personable and quicker. Customer service chatbots can interact with customers based on the language of their input text and respond accordingly. If a customer needs help with a product, the chatbot server can field the corresponding language product manual, or transfer it to a call center for the specific language.

The following is a list of some of the supported languages:

Table 2: Supported Languages

Afrikaans	Albanian	Arabic	Armenian	Azerbaijani	Basque
Belarusian	Bengali	Bosnian	Bulgarian	Burmese	Cantonese
Catalan	Cebuano	Chinese	Croatian	Czech	Danish
Dutch	Eastern Punjabi	Egyptian Arabic	English	Esperanto	Estonian
Finnish	French	Georgian	German	Greek	Hebrew
Hindi	Hungarian	Icelandic	Indonesian	Irish	Italian
Japanese	Javanese	Kannada	Kazakh	Korean	Kurdish (Sorani)
Latin	Latvian	Lithuanian	Macedonian	Malay	Malayalam
Marathi	Minangkabau	Nepali	Norwegian (Bokmal)	Norwegian (Nynorsk)	Persian
Polish	Portuguese	Romanian	Russian	Serbian	Serbo-Croatian
Slovak	Slovene	Spanish	Swahili	Swedish	Tagalog
Tamil	Telugu	Thai	Turkish	Ukrainian	Urdu
Uzbek	Vietnamese	Welsh			

The next example determines the language of the text, the [ISO 639-1](#) language code, and a probability score.

```
s.language_dominant
```

	Code	Language	Score
0	en	English	0.999678

Named Entity Recognition

Named entity recognition (NER) detects named entities in text. The NER model uses NLP, which uses machine learning to find predefined named entities. This model also provides a confidence score for each entity and is a value from 0 to 1. The returned data is the text of the entity, its position in the document, and its length. It also identifies the type of entity, a probability score that it is an entity of the stated type.

The following are the supported entity types:

- **DATE:** Absolute or relative dates, periods, and date range.
- **EMAIL:** Email address.
- **EVENT:** Named hurricanes, sports events, and so on.
- **FAC:** Facilities; Buildings, airports, highways, bridges, and so on.

- GPE: Geopolitical entity; Countries, cities, and states.
- IPADDRESS: IP address according to IPv4 and IPv6 standards.
- LANGUAGE: Any named language.
- LOCATION: Non-GPE locations, mountain ranges, and bodies of water.
- MONEY: Monetary values, including the unit.
- NORP: Nationalities, religious, and political groups.
- ORG: Organization; Companies, agencies, institutions, and so on.
- PERCENT: Percentage.
- PERSON: People, including fictional characters.
- PHONE_NUMBER: Supported phone numbers.
 - (“GB”) - United Kingdom
 - (“AU”) - Australia
 - (“NZ”) - New Zealand
 - (“SG”) - Singapore
 - (“IN”) - India
 - (“US”) - United States
- PRODUCT: Vehicles, tools, foods, and so on (not services).
- QUANTITY: Measurements, as weight or distance.
- TIME: Anything less than 24 hours (time, duration, and so on).
- URL: URL

The following example lists the named entities:

s . ner

The output gives the named entity, its location, and offset position in the text. It also gives a probability and score that this text is actually a named entity along with the type.

	PII	Length	Offset	Score	Entity	Type
0	True	23	0	1.0	Lawrence Joseph Ellison	PERSON
1	False	15	30	1.0	August 17, 1944	DATE
2	False	18	215	1.0	Oracle Corporation	ORG
3	False	12	241	1.0	October 2019	DATE
4	False	6	284	1.0	Forbes	ORG
5	False	13	339	1.0	United States	GPE
6	False	13	425	1.0	\$69.1 billion	MONEY
7	False	13	467	1.0	\$54.5 billion	MONEY
8	False	8	484	1.0	2018.[4]	DATE
9	False	13	560	1.0	United States	GPE
10	False	5	575	1.0	Lanai	GPE
11	False	16	588	1.0	Hawaiian Islands	LOCATION
12	False	4	648	1.0	3000	CARDINAL

Text Classification

Text classification analyses the text and identifies categories for the content with a confidence score. Text classification uses NLP techniques to find insights from textual data. It returns a category from a set of predefined categories. This text classification uses NLP and relies on the main objective lies on zero-shot learning. It classifies text with no or minimal data to train. The content of a collection of documents is analyzed to determine common themes.

The next example classifies the text and gives a probability score that the text is in that category.

```
s.text_classification
```

	Label	Score
0	Finance/Investing	0.369175

9.7.1.5 RegEx Match

Text documents are often parsed looking for specific patterns to extract information like emails, dates, times, web links, and so on. This pattern matching is often done using RegEx, which is hard to write, modify, and understand. Custom written RegEx often misses the edge cases. `ADSSString` provides a number of common RegEx patterns so that your work is simplified. You can use the following patterns:

- `credit_card`: Credit card number.
- `dates`: Dates in a variety of standard formats.
- `email`: Email address.
- `ip`: IP addresses, versions IPV4 and IPV6.
- `link`: Text that appears to be a link to a website.
- `phone_number_US`: USA phone numbers including those with extensions.
- `price`: Text that appears to be a price.
- `ssn`: USA social security number.
- `street_address`: Street address.
- `time`: Text that appears to be a time and less than 24 hours.
- `zip_code`: USA zip code.

The preceding `ADSSString` properties return an array with each pattern that in matches. The following examples demonstrate how to extract email addresses, dates ,and links from the text. Note that the text is extracted as is. For example, the dates aren't converted to a standard format. The returned value is the text as it is represented in the input text. Use the `datetime.strptime()` method to convert the date to a date time stamp.

```
s = ADSSString("Get in touch with my associates john.smith@example.com and jane.
↪johnson@example.com to schedule")
s.email
```

```
['john.smith@example.com', 'jane.johnson@example.com']
```

```
s = ADSSString("She is born on Jan. 19th, 2014 and died 2021-09-10")
s.date
```

```
['Jan. 19th, 2014', '2021-09-10']
```

```
s = ADSSString("Follow the link www.oracle.com to Oracle's homepage.")
s.link
```

```
['www.oracle.com']
```

9.7.1.6 Still a String

While `ADSString` expands your feature engineering capabilities, it can still be treated as a `str` object. Any standard operation on `str` is preserved in `ADSString`. For instance, you can convert it to lowercase:

```
hello_world = "HELLO WORLD"
s = ADSString(hello_world)
s.lower()
```

```
'hello world'
```

You could split a text string.

```
s.split()
```

```
['HELLO', 'WORLD']
```

You can use all the `str` methods, such as the `.replace()` method, to replace text.

```
s.replace("L", "N")
```

```
'HENNO WORND'
```

You can perform a number of `str` manipulation operations, such as `.lower()` and `.upper()` to get an `ADSString` object back.

```
isinstance(s.lower().upper(), ADSString)
```

```
True
```

While a new `ADSString` object is created with `str` manipulation operations, the equality operation holds.

```
s.lower().upper() == s
```

```
True
```

The equality operation even holds between `ADSString` objects (`s`) and `str` objects (`hello_world`).

```
s == hello_world
```

```
True
```

9.7.2 Text Extraction

Convert files such as PDF, and Microsoft Word files into plain text. The data is stored in Pandas dataframes and therefore it can easily be manipulated and saved. The text extraction module allows you to read files of various file formats, and convert them into different formats that can be used for text manipulation. The most common `DataLoader` commands are demonstrated, and some advanced features, such as defining custom backend and file processor.

```
import ads
import fsspec
```

(continues on next page)

(continued from previous page)

```
import oci
import os
import pandas as pd
import shutil
import time
import tempfile

from ads.text_dataset.backends import Base
from ads.text_dataset.dataset import TextDatasetFactory as textfactory
from ads.text_dataset.extractor import FileProcessor, FileProcessorFactory
from ads.text_dataset.options import Options
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

ads.set_debug_mode()
ads.set_auth("resource_principal")
```

9.7.2.1 Introduction

Text extraction is the process of extracting text from one document and converting it into another form, typically plain text. For example, you can extract the body of text from a PDF document that has figures, tables, images, and text. The process can also be used to extract metadata about the document. Generally, text extraction takes a corpus of documents and returns the extracted text in a structured format. In the ADS text extraction module, that format is a Pandas dataframe.

The Pandas dataframe has a record in each row. That record can be an entire document, a sentence, a line of text, or some other unit of text. In the examples, you explore using a row to indicate a line of text and an entire document.

The ADS text extraction module supports:

- Input formats: `text`, `pdf` and `docx` or `doc`.
- Output formats: Use `pandas` for Pandas dataframe, or `cudf` for a cuDF dataframe.
- Backends: [Apache Tika](#) (default) and [pdfplumber](#) (for PDF).
- Source location: local block volume, and in cloud storage such as the Oracle Cloud Infrastructure (OCI) Object Storage.
- Options to extract metadata.

You can manipulate files through the `DataLoader` object. Some of the most common commands are:

- `.convert_to_text()`: Convert document to text and then save them as plain text files.
- `.metadata_all()` and `.metadata_schema()`: Extract metadata from each file.
- `.read_line()`: Read files line-by-line. Each line corresponds to a record in the corpus.
- `.read_text()`: Read files where each file corresponds to a record in the corpus.

9.7.2.1.1 Configure the Data Source

The OCI Data Science service has a corpus of text documents that are used in the examples. This corpus is stored in a publicly accessible OCI Object Storage bucket. The following variables define the Object Storage namespace and the bucket name. You can update these variables to point at your Object Storage bucket, but you might also have to change some of the code in the examples so that the keys are correct.

```
namespace = 'bigdatadatasciencelarge'
bucket = 'hosted-ds-datasets'
```

9.7.2.2 Load

The `TextDatasetFactory`, which is aliased to `textfactory` in this notebook, provides access to the `DataLoader`, and `FileProcessor` objects. The `DataLoader` is a file format-specific object for reading in documents such as PDF and Word documents. Internally, a data loader binds together a file system interface (in this case `fsspec`) for opening files. The `FileProcessor` object is used to convert these files into plain text. It also has an `engine` object to control the output format. For a given `DataLoader` object, you can customize both the `FileProcessor` and `engine`.

Generally, the first step in reading a corpus of documents is to obtain a `DataLoader` object. For example, `TextDatasetFactory.format('pdf')` returns a `DataLoader` for PDFs. Likewise, you can get a Word document loaders by passing in `docx` or `doc`. You can choose an engine that controls how the data is returned. The default engine is a Python generator. If you want to use the data as a dataframe, then use the `.engine()` method. A call to `.engine('pandas')` returns the data as a Pandas dataframe. On a GPU machine, you can use cuDF dataframes with a call to `.engine('cudf')`.

The `.format()` method controls the backend with `Apache Tika` and `pdfplumber` being builtin. In addition, you can write your own backend and plug it into the system. This allows you complete control over the backend. The file processor is used to actually process a specific file format.

To obtain a `DataLoader` object, call the use the `.format()` method on `textfactory`. This returns a `DataLoader` object that can then be configured with the `.backend()`, `.engine()`, and `.options()` methods. The `.backend()` method is used to define which backend is to manage the process of parsing the corpus. If this is not specified then a sensible default backend is chosen based on the file format that is being processed. The `.engine()` method is used to control the output format of the data. If it is not specified, then an iterator is returned. The `.options()` method is used to add extra fields to each record. These would be things such as the filename, or metadata about the file. There are more details about this and the other configuration methods in the examples.

9.7.2.2.1 Read a Dataset

In this example you create a `DataLoader` object by calling `textfactory.format('pdf')`. This `DataLoader` object is configured to read PDF documents. You then change the backend to use `pdfplumber` with the method `.backend('pdfplumber')`. It's easier to work with the results if they are in a dataframe. So, the method `.engine('pandas')` returns a Pandas dataframe.

After you have the `DataLoader` object configured, you process the corpus. In this example, the corpus is a single PDF file. It is read from a publicly accessible OCI Object Storage bucket. The `.read_line()` method is used to read in the corpus where each line of the document is treated as a record. Thus, each row in the returned dataframe is a line of text from the corpus.

```
dl = textfactory.format('pdf').backend('pdfplumber').engine('pandas')
df = dl.read_line(
    f'oci://{bucket}/{namespace}/pdf_sample/paper-0.pdf',
    storage_options={"config": {}},
```

(continues on next page)

(continued from previous page)

```
)
df.head()
```

0

```
0      PREVENTING CHRONIC DISEASE\n
1  PUBLICHEALTHRESEARCH, ...
2      Volume 15, E97 ...
3      \n
4      ORIGINAL RESEARCH\n
```

9.7.2.2.2 Read Options

Typically, you want to treat each line of a document or each document as a record. The method `.read_line()` processes a corpus, and return each line in the documents as a text string. The method `.read_text()` treats each document in the corpus as a record.

Both the `.read_line()` and `.read_text()` methods parse the corpus, convert it to text, and reads it into memory. The `.convert_to_text()` method does the same processing as `.read_text()`, but it outputs the plain text to files. This allows you to post-process the data without having to *again* convert the raw documents into plain text documents, which can be an expensive process.

Each document can have a custom set of metadata that describes the document. The `.metadata_all()` and `.metadata_schema()` methods allow you to access this metadata. Metadata is represented as a key-value pair. The `.metadata_all()` returns a set of key-value pairs for each document. The `.metadata_schema()` returns what keys are used in defining the metadata. This can vary from document to document and this method creates a list of all observed keys. You use this to understand what metadata is available in the corpus.

`.read_line()`

The `.read_line()` method allows you to read a corpus line-by-line. In other words, each line in a file corresponds to one record. The only required argument to this method is `path`. It sets the path to the corpus, and it can contain a glob pattern. For example, `oci://{bucket}@{namespace}/pdf_sample/**/*.pdf`, `oci://{bucket}@{namespace}/20news-small/**/[1-9]*`, or `/home/datascience/<path-to-folder>/[A-Za-z]*.docx` are all valid paths that contain a glob pattern for selecting multiple files. The `path` parameter can also be a list of paths. This allows for reading files from different file paths.

The optional parameter `udf` stands for a user-defined function. This parameter can be a callable Python object, or a regular expression (RegEx). If it is a callable Python object, then the function must accept a string as an argument and returns a tuple. If the parameter is a RegEx, then the returned values are the captured RegEx patterns. If there is no match, then the record is ignored. This is a convenient method to selectively capture text from a corpus. In either case, the `udf` is applied on the record level, and is a powerful tool for data transformation and filtering.

The `.read_line()` method has the following arguments:

- `df_args`: Arguments to pass to the engine. It only applies to Pandas and cuDF dataframes.
- `n_lines_per_file`: Maximal number of lines to read from a single file.
- `path`: The path to the corpus.
- `storage_options`: Options that are necessary for connecting to OCI Object Storage.
- `total_lines`: Maximal number of lines to read from all files.

- **udf**: User-defined function for data transformation and filtering.

Examples

Python Callable udf

In the next example, a lambda function is used to create a Python callable object that is passed to the `udf` parameter. The lambda function takes a line and splits it based on white space to tokens. It then counts the number of tokens, and returns a tuple where the first element is the token count and the second element is the line itself.

The `df_args` parameter is used to change the column names into user-friendly values.

```
dl = textfactory.format('docx').engine('pandas')
df = dl.read_line(
    path=f'oci://{bucket}@{namespace}/docx_sample/*.docx',
    udf=lambda x: (len(x.strip().split()), x),
    storage_options={"config": {}},
    df_args={'columns': ['token count', 'text']},
)
df.head()
```

token count		text
0	1	notes
1	0	
2	2	Geography Proper
3	94	Generally, geographers before the 70s were con...
4	100	A great example of this is Cuba - think of it ...

Regular Expression udf

In this example, the corpus is a collection of log files. A RegEx is used to parse the standard Apache log format. If a line does not match the pattern, it is discarded. If it does match the pattern, then a tuple is returned where each element is a value in the RegEx `capture group`.

This example uses the default engine, which returns an iterator. The `next()` method is used to iterate through the values.

```

APACHE_LOG_PATTERN = r'^[(\S+)\s(\S+)\s(\d+)\s(\d+:\d+:\d+)\s(\d+)]\s(\S+)\s(\S+)\s(\S+)\s(\S+)'
dl = textfactory.format('txt')
df = dl.read_line(
    f'oci://{bucket}@{namespace}/log_sample/*.log',
    udf=APACHE_LOG_PATTERN,
    storage_options={"config": {}},
)
next(df)

```

```
['Sun',
 'Dec',
 '04',
 '04:47:44',
 '2005',
 '[notice]',
 'workerEnv.init()',
 'ok',
 '/etc/httpd/conf/workers2.properties']
```

```.read_text()```

If you want to treat each document in a corpus as a record, use the `.read_text()` method. The `path` parameter is the only required parameter as it defines the location of the corpus.

The optional `udf` parameter stands for a user-defined function. This parameter can be a callable Python object or a `Regex`.

The `.read_text()` method has the following arguments:

- `df_args`: Arguments to pass to the engine. It only applies to Pandas and cuDF dataframes.
- `path`: The path to the corpus.
- `storage_options`: Options that are necessary for connecting to OCI Object Storage.
- `total_files`: The maximum number of files that should be processed.
- `udf`: User-defined function for data transformation and filtering.

## Examples

### `total_files`

In this example, there are six files in the corpus. However, the `total_files` parameter is set to 4 so only the first four files are processed. There is no guarantee which four will actually be processed. However, this parameter is commonly used to limit the size of the data when you are developing the code for the model. Later on, it is often removed so the entire corpus is processed.

This example also demonstrates the use of a list, plus globbing, to define the corpus. Notice that the `path` parameter is a list with two file paths. The output shows the dataframe has four rows and so only four files were processed.

```
dl = textfactory.format('docx').engine('pandas')
df = dl.read_text(
 path=[f'oci://{bucket}@{namespace}/docx_sample/*.docx', f'oci://{bucket}@{namespace}/
 ↪ docx_sample/*.doc'],
 total_files=4,
 storage_options={"config": {}},
)
df.shape
```

```
(4, 1)
```

## `.convert_to_text()`

Converting a set of raw documents can be an expensive process. The `.convert_to_text()` method allows you to convert a corpus of source documents and write them out as plain text files. Each document input document is written to a separate file that has the same name as the source file. However, the file extension is changed to `.txt`. Converting the raw documents allows you to post-process the raw text multiple times while only have to convert it once.

The `src_path` parameter defines the location of the corpus. The `dst_path` parameter gives the location where the plain text files are to be written. It can be an Object Storage bucket or the local block storage. If the directory does not exist, it is created. It overwrites any files in the directory.

The `.convert_to_text()` method has the following arguments:

- `dst_path`: Object Storage or local block storage path where plain text files are written.
- `encoding`: Encoding for files. The default is `utf-8`.
- `src_path`: The path to the corpus.
- `storage_options`: Options that are necessary for connecting to Object Storage.

The following example converts a corpus, and writes it to a temporary directory. It then lists all the plain text files that were created in the conversion process.

```
dst_path = tempfile.mkdtemp()
dl = textfactory.format('pdf')
dl.convert_to_text(
 src_path=f'oci://{bucket}@{namespace}/pdf_sample/*.pdf',
 dst_path=dst_path,
 storage_options={"config": {}},
)
print(os.listdir(dst_path))
shutil.rmtree(dst_path)
```

```
['paper-2.txt', 'paper-0.txt', 'Emerging Infectious Diseases copyright info.txt',
↳ 'Preventing Chronic Disease Copyright License.txt', 'Budapest Open Access Initiative _u
↳ Budapest Open Access Initiative.txt', 'paper-1.txt']
```

Each document can contain metadata. The purpose of the `.metadata_all()` method is to capture this information for each document in the corpus. There is no standard set of metadata across all documents so each document could return different set of values.

The `path` parameter is the only required parameter as it defines the location of the corpus.

The `.metadata_all()` method has the following arguments:

- `encoding`: Encoding for files. The default is `utf-8`.
- `path`: The path to the corpus.
- `storage_options`: Options that are necessary for connecting to Object Storage.

The next example processes a corpus of PDF documents using `pdfplumber`, and prints the metadata for the first document.

```
dl = textfactory.format('pdf').backend('pdfplumber').option(Options.FILE_NAME)
metadata = dl.metadata_all(
 path=f'oci://{bucket}@{namespace}/pdf_sample/Emerging Infectious Diseases copyright_
↳ info.pdf',
```

(continues on next page)

(continued from previous page)

```

 storage_options={"config": {}}
)
next(metadata)

```

```

{'Creator': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
↳ like Gecko) Chrome/91.0.4472.114 Safari/537.36',
'Producer': 'Skia/PDF m91',
'CreationDate': 'D:20210802234012+00'00'',
'ModDate': 'D:20210802234012+00'00''}

```

The backend that is used can affect what metadata is returned. For example, the Tika backend returns more metadata than pdfplumber, and also the names of the metadata elements are also different. The following example processes the same PDF document as previously used, but you can see that there is a difference in the metadata.

```

dl = textfactory.format('pdf').backend('default')
metadata = dl.metadata_all(
 path=f'oci://{bucket}/{namespace}/pdf_sample/Emerging Infectious Diseases copyright_
↳ info.pdf',
 storage_options={"config": {}}
)
next(metadata)

```

```

{'Content-Type': 'application/pdf',
'Creation-Date': '2021-08-02T23:40:12Z',
'Last-Modified': '2021-08-02T23:40:12Z',
'Last-Save-Date': '2021-08-02T23:40:12Z',
'X-Parsed-By': ['org.apache.tika.parser.DefaultParser',
'org.apache.tika.parser.pdf.PDFParser'],
'access_permission:assemble_document': 'true',
'access_permission:can_modify': 'true',
'access_permission:can_print': 'true',
'access_permission:can_print_degraded': 'true',
'access_permission:extract_content': 'true',
'access_permission:extract_for_accessibility': 'true',
'access_permission:fill_in_form': 'true',
'access_permission:modify_annotations': 'true',
'created': '2021-08-02T23:40:12Z',
'date': '2021-08-02T23:40:12Z',
'dc:format': 'application/pdf; version=1.4',
'dcterms:created': '2021-08-02T23:40:12Z',
'dcterms:modified': '2021-08-02T23:40:12Z',
'meta:creation-date': '2021-08-02T23:40:12Z',
'meta:save-date': '2021-08-02T23:40:12Z',
'modified': '2021-08-02T23:40:12Z',
'pdf:PDFVersion': '1.4',
'pdf:charsPerPage': '2660',
'pdf:docinfo:created': '2021-08-02T23:40:12Z',
'pdf:docinfo:creator_tool': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
↳ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36',
'pdf:docinfo:modified': '2021-08-02T23:40:12Z',
'pdf:docinfo:producer': 'Skia/PDF m91',
'pdf:encrypted': 'false',

```

(continues on next page)

(continued from previous page)

```
'pdf:hasMarkedContent': 'true',
'pdf:hasXFA': 'false',
'pdf:hasXMP': 'false',
'pdf:unmappedUnicodeCharsPerPage': '0',
'producer': 'Skia/PDF m91',
'xmp:CreatorTool': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36_
↪(KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36',
'xmpTPg:NPages': '1'}
```

**``.metadata\_schema()``**

As briefly discussed in the `.metadata_all()` method section, there is no standard set of metadata across all documents. The `.metadata_schema()` method is a convenience method that returns what metadata is available in the corpus. It returns a list of all observed metadata fields in the corpus. Since each document can have a different set of metadata, all the values returned may not exist in all documents. It should also be noted that the engine used can return different metadata for the same document.

The `path` parameter is the only required parameter as it defines the location of the corpus.

Often, you don't want to process an entire corpus of documents to get a sense of what metadata is available. Generally, the engine returns a fairly consistent set of metadata. The `n_files` option is handy because it limits the number of files that are processed.

The `.metadata_schema()` method has the following arguments:

- `encoding`: Encoding for files. The default is `utf-8`.
- `n_files`: Maximum number of files to process. The default is 1.
- `path`: The path to the corpus.
- `storage_options`: Options that are necessary for connecting to Object Storage.

The following example uses the `.metadata_schema()` method to collect the metadata fields on the first two files in the corpus. The `n_files=2` parameter is used to control the number of files that are processed.

```
dl = textfactory.format('pdf').backend('pdfplumber')
schema = dl.metadata_schema(
 f'oci://{bucket}/{namespace}/pdf_sample/*.pdf',
 storage_options={"config": {}},
 n_files=2
)
print(schema)
```

```
['ModDate', 'Producer', 'CreationDate', 'Creator']
```

**9.7.2.3 Augment Records**

The `text_dataset` module has the ability to augment the returned records with additional information using the `.option()` method. This method takes an enum from the `Options` class. The `.option()` method can be used multiple times on the same `DataLoader` to select a set of additional information that is returned. The `Options.FILE_NAME` enum returns the filename that is associated with the record. The `Options.FILE_METADATA` enum allows you to extract individual values from the document's metadata. Notice that the engine used can return different metadata for the same document.

### 9.7.2.3.1 Examples

#### ``Options.FILE\_NAME``

The following example uses `.option(Options.FILE_NAME)` to augment to add the filename of each record that is returned. The example uses the `txt` for the `FileProcessor`, and `Tika` for the backend. The engine is `Pandas` so a dataframe is returned. The `df_args` option is used to rename the columns of the dataframe. Notice that the returned dataframe has a column named `path`. This is the information that was added to the record from the `.option(Options.FILE_NAME)` method.

```
dl = textfactory.format('txt').backend('tika').engine('pandas').option(Options.FILE_NAME)
df = dl.read_text(
 path=f'oci://{bucket}@{namespace}/20news-small/**/[1-9]*',
 storage_options={"config": {}},
 df_args={'columns': ['path', 'text']}
)
df.head()
```

	path	text
0	hosted-ds-datasets@bigdatadatasciencelarge/20n...	\tThe Orioles' pitching staff again is having ...
1	hosted-ds-datasets@bigdatadatasciencelarge/20n...	Subject: Re: Eck vs Rickey (was Re: Rickey's w...
2	hosted-ds-datasets@bigdatadatasciencelarge/20n...	Hell, the Orioles' Opening Day game could easi...
3	hosted-ds-datasets@bigdatadatasciencelarge/20n...	There's a lot of whining about how much player...
4	hosted-ds-datasets@bigdatadatasciencelarge/20n...	In article <1993Apr5.173500.26383@ra.msstate.e...

#### ``Options.FILE\_METADATA``

You can add metadata about a document to a record using `.option(Options.FILE_METADATA, {'extract': ['<key1>', '<key2>']})`. When using `Options.FILE_METADATA`, there is a required second parameter. It takes a dictionary where the key is the action to be taken. In the next example, the `extract` key provides a list of metadata that can be extracted. When a list is used, the returned value is also a list of the metadata values. The example uses repeated calls to `.option()` where different metadata values are extracted. In this case, a list is not returned, but each value is in a separate `Pandas` column.

```
dl = textfactory.format('docx').engine('pandas') \
 .option(Options.FILE_METADATA, {'extract': ['Character Count']}) \
 .option(Options.FILE_METADATA, {'extract': ['Paragraph-Count']}) \
 .option(Options.FILE_METADATA, {'extract': ['Author']})
df = dl.read_text(
 path=f'oci://{bucket}@{namespace}/docx_sample/*.docx',
 storage_options={"config": {}},
 df_args={'columns': ['character count', 'paragraph count', 'author', 'content']},
)
df.head()
```

	character count	paragraph count	author	content
0	[444461]	[1042]	[miked_000]	notes\n\nGeography Proper\nGenerally, geograph...
1	[444461]	[1042]	[miked_000]	notes\n\nGeography Proper\nGenerally, geograph...
2	[119218]	[279]	[Miranda, Team 2012]	***The Gift K***\nNotes\n\nRelation to Colonia...

#### 9.7.2.4 Custom File Processor and Backend

The `text_dataset` module supports a number of file processors and backends. However, it isn't practical to provide these for all possible documents. So, the `text_dataset` allows you to create your own.

When creating a custom file processor, you must register it with ADS using the `FileProcessorFactory.register()` method. The first parameter is the name that you want to associate with the file processor. The second parameter is the class that is to be registered. There is no need to register the backend class.

##### 9.7.2.4.1 Custom Backend

To create a backend, you need to develop a class that inherits from the `ads.text_dataset.backends.Base` class. In your class, you need to overload any of the following methods that you want to use with: `.read_line()`, `.read_text()`, `.convert_to_text()`, and `.get_metadata()`. The `.get_metadata()` method must be overload if you want to use the `.metadata_all()` and `.metadata_schema()` methods in your backend.

The `.convert_to_text()` method takes a file handler, destination path, filename, and storage options as parameters. This method must write the plain text file to the destination path, and return the path of the file.

The `.get_metadata()` method takes a file handler as an input parameter, and returns a dictionary of the metadata. The `.metadata_all()` and `.metadata_schema()` methods don't need to be overload because they use the `.get_metadata()` method to return their results.

The `.read_line()` method must take a file handle, and have a `yield` statement that returns a plain text line from the document.

The `.read_text()` method has the same requirements as the `.read_line()` method, except it must `yield` the entire document as plain text.

The following are the method signatures:

```
convert_to_text(self, fhandler, dst_path, fname, storage_options)
get_metadata(self, fhandler)
read_line(self, fhandler)
read_text(self, fhandler)
```

##### 9.7.2.4.2 Custom File Processor

To create a custom file processor you must develop a class that inherits from `ads.text_dataset.extractor.FileProcessor`. Generally, there are no methods that need to be overloaded. However, the `backend_map` class variable has to be defined. This is a dictionary where the key is the name of the format that it supports and the value is the file processor class. There must be a key called `default` that is used when no file processor is defined for the `DataLoader`. An example of the `backend_map` is:

```
backend_map = {'default': MyCustomBackend, 'tika': Tika, 'custom': MyCustomBackend}
```

### 9.7.2.4.3 Example

In the next example, you create a custom backend class called `ReverseBackend`. It overloads the `.read_line()` and `.read_text()` methods. This toy backend returns the records in reverse order.

The `TextReverseFileProcessor` class is used to create a new file processor for use with the backend. This class has the `backend_map` class variable that maps the backend label to the backend object. In this case, the only format that is provided is the default class.

Having defined the backend (`TextReverseBackend`) and file processor (`TextReverseFileProcessor`) classes, the format must be registered. You register it with the `FileProcessorFactory.register('text_reverse', TextReverseFileProcessor)` command where the first parameter is the format and the second parameter is the file processor class.

```
class TextReverseBackend(Base):
 def read_line(self, fhandler):
 with fhandler as f:
 for line in f:
 yield line.decode()[::-1]

 def read_text(self, fhandler):
 with fhandler as f:
 yield f.read().decode()[::-1]

class TextReverseFileProcessor(FileProcessor):
 backend_map = {'default': TextReverseBackend}

FileProcessorFactory.register('text_reverse', TextReverseFileProcessor)
```

Having created the custom backend and file processor, you use the `.read_line()` method to read in one record and print it.

```
dl = textfactory.format('text_reverse')
reverse_text = dl.read_line(
 f'oci://{bucket}@{namespace}/20news-small/rec.sport.baseball/100521',
 total_lines=1,
 storage_options={"config": {}},
)
text = next(reverse_text)[0]
print(text)
```

```
)uiL C evetS(ude.uhj.fch.xinuhj@larimda :morF
```

The `.read_line()` method in the `TextReverseBackend` class reversed the characters in each line of text that is processed. You can confirm this by reversing it back.

```
text[::-1]
```

```
'From: admiral@jhunix.hcf.jhu.edu (Steve C Liu)n'
```



## VISUALIZE DATA

Data visualization is an important aspect of data exploration, analysis, and communication. Generally, visualization of the data is one of the first steps in any analysis. It allows the analysts to efficiently gain an understanding of the data and guides the exploratory data analysis (EDA) and the modeling process.

An efficient and flexible data visualization tool can provide a lot of insight into the data. ADS provides a smart visualization tool. It automatically detects the data type and renders plots that optimally represent the characteristics of the data. Within ADS, custom visualizations can be created using any plotting library.

### 10.1 Automatic

The ADS `show_in_notebook()` method creates a comprehensive preview of all the basic information about a dataset including:

- The predictive data type (for example, regression, binary classification, or multinomial classification).
- The number of columns and rows.
- Feature type information.
- Summary visualization of each feature.
- The correlation map.
- Any warnings about data conditions that you should be aware of.

To improve plotting performance, the ADS `show_in_notebook()` method uses an optimized subset of the data. This smart sample is selected so that it is statistically representative of the full dataset. The correlation map is only displayed when the data only has numerical (`continuous` or `ordinal`) columns.

```
ds.show_in_notebook()
```

To visualize the correlation, call the `show_corr()` method. If the correlation matrices have not been cached, this call triggers the `corr()` function which calculates the correlation matrices.

`corr()` uses the following methods to calculate the correlation based on the data types:

- Continuous-Continuous: ``Pearson`` method [https://en.wikipedia.org/wiki/Pearson\\_correlation\\_coefficient](https://en.wikipedia.org/wiki/Pearson_correlation_coefficient)>`\_\_`. The correlations range from -1 to 1.
- Categorical-Categorical: ``Cramer's V`` method [https://en.wikipedia.org/wiki/Cram%C3%A9r's\\_V](https://en.wikipedia.org/wiki/Cram%C3%A9r's_V)>`\_\_`. The correlations range from 0 to 1.
- Continuous-Categorical: ``Correlation Ratio`` method [https://en.wikipedia.org/wiki/Correlation\\_ratio](https://en.wikipedia.org/wiki/Correlation_ratio)>`\_\_`. The correlations range from 0 to 1.

▼ Summary

Name: DataFrame from oracle\_classification\_dataset1\_150K.csv

Type: BinaryClassificationDataset

150,000 Rows, 49 Columns

Column Types:

- continuous: 39 features
- categorical: 10 features

Note: Visualizations use a sampled subset of the dataset, this is to improve plotting performance. The sample size is calculated to be statistically significant within the confidence level: 95 and confidence interval: 1.0. The sampled data has 10,000 rows

- The confidence level refers to the long-term success rate of the method, that is, how often this type of interval will capture the parameter of interest.
- A specific confidence interval gives a range of plausible values for the parameter of interest

► Features (49)

► Correlations

► Warnings (3)

▼ Features (49)

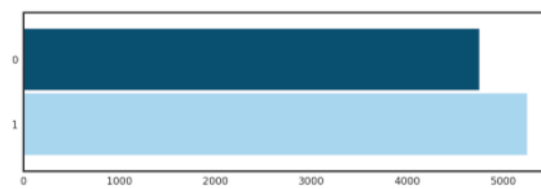
Note these are computed on the entire dataset.

	count	mean	std	min	25%	50%	75%	max	missing	skew
class	150000	0.53	0.5	0	0	1	1	1	0	-0.11541953
col01	150000	0.01	3.68	-16.22	-2.47	0.01	2.47	16.48	0	-0.001944536
col02	150000	10.1	0.3	10	10	10	10	11	0	2.670402
col03	150000	0	2.21	-10.52	-1.48	0	1.5	9.94	0	0.0021300788
col04	150000	0.79	200.6	-889.09	-134.85	0.22	136.36	1124.37	0	0.0021092156
col05	150000	-0	0.12	-0.68	-0.05	-0	0.05	0.7	0	-0.022465346
col06	150000	-0.01	3.01	-15.08	-2.01	-0.01	2.01	14.9	0	-0.0033007577
col07	150000	-9.3	0.46	-10	-10	-9	-9	-9	0	-0.87149529
col08	150000	100.9	0.3	100	101	101	101	101	0	-2.6545245
col09	150000	-9.3	0.46	-10	-10	-9	-9	-9	0	-0.87139146
col010	150000	-0	0.4	-1.65	-0.27	-0	0.27	1.83	0	0.00080830709
col011	150000	1000.99	1.4	1000	1000	1000	1003.1	1003.1	0	0.77117106
col012	150000	10.2	0.4	10	10	10	10	11	0	1.4987655

## Feature Visualizations...

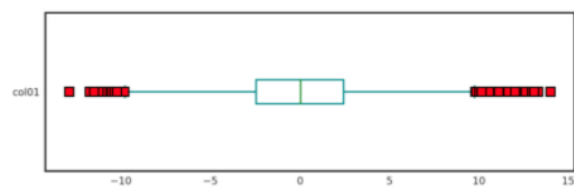
class

- type: categorical (int64)
- missing\_percentage: 0.0%
- categorical statistics:
  - unique percentage: 0.020%
  - mode: 1
  - count: 10,000
  - unique: 2
  - top: 1
  - freq: 5,249



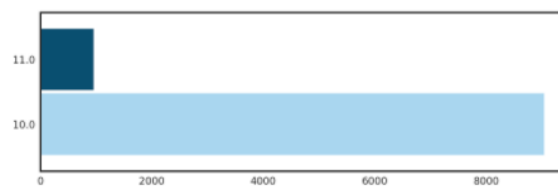
col01

- type: continuous (float64)
- missing\_percentage: 0.0%
- continuous statistics:
  - mode: -12.888
  - median: 0.049
  - kurtosis: 0.064
  - variance: 13.632
  - skewness: 0.051
  - outlier\_percentage: 0.220%
- count: 10,000
- mean: 0.050
- std: 3.692
- min: -12.888
- 25%: -2.440
- 50%: 0.049
- 75%: 2.454
- max: 13.991



col02

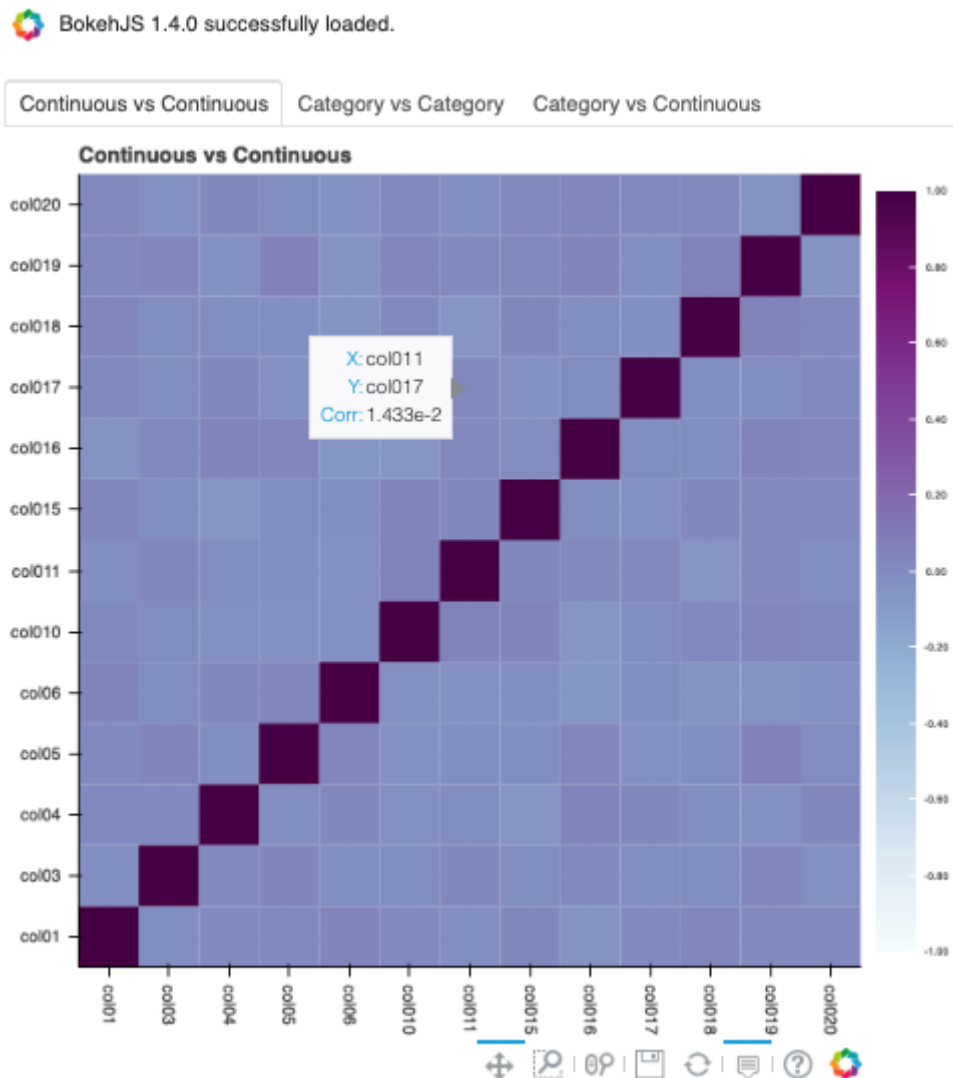
- type: categorical (float64)
- missing\_percentage: 0.0%
- categorical statistics:
  - unique percentage: 0.020%
  - mode: 10
  - count: 10,000
  - unique: 2
  - top: 10
  - freq: 9,043



Correlations are displayed independently because the correlations are calculated using different methodologies and the ranges are not the same. Consolidating them into one matrix could be confusing and inconsistent.


**Note:** Continuous features consist of continuous and ordinal types. Categorical features consist of categorical and zipcode types.

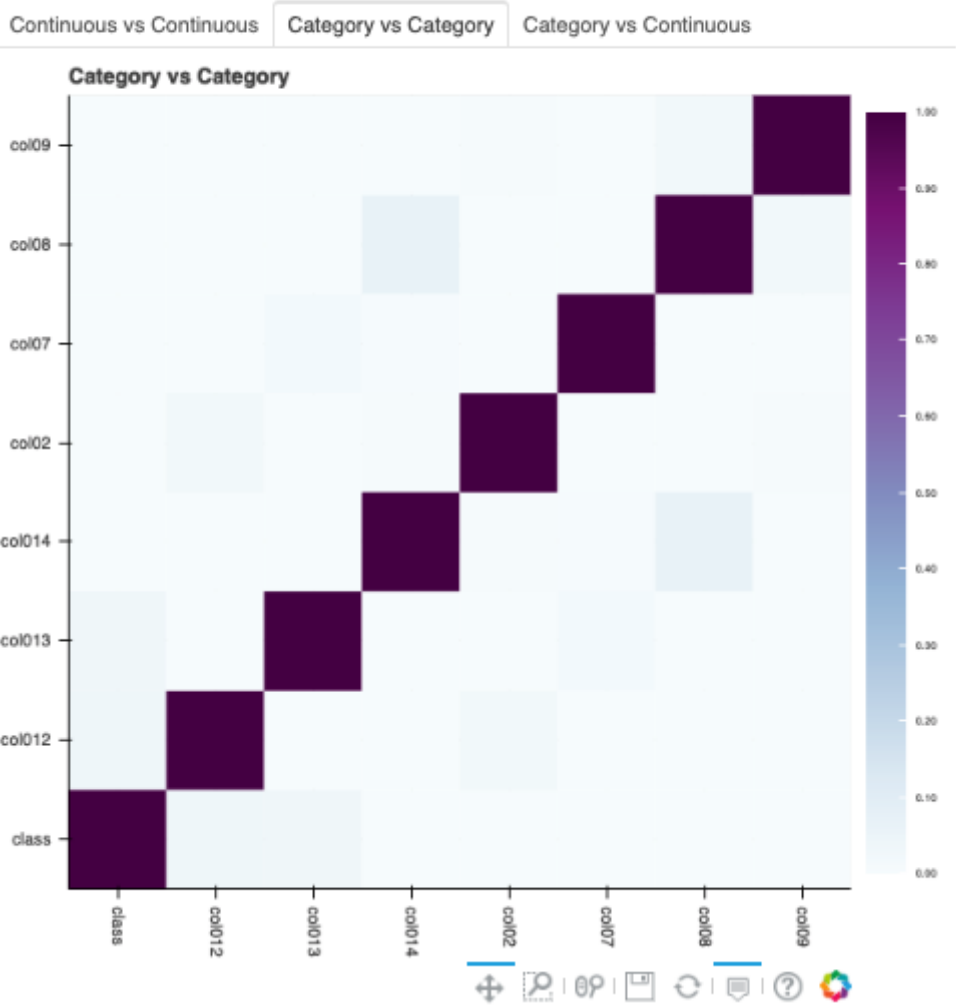
```
ds.show_corr(nan_threshold=0.8, correlation_methods='all')
```



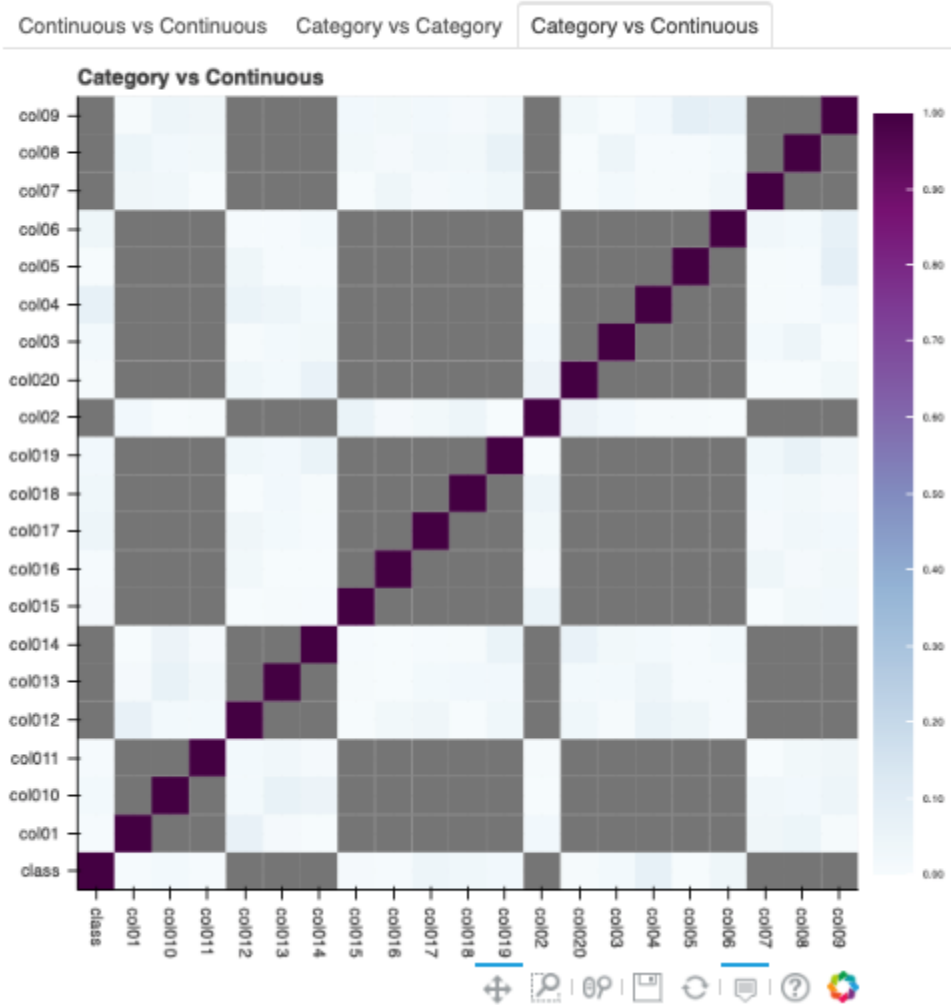
By default, `nan_threshold` is set to 0.8. This means that if more than 80% of the values in a column are missing, that column is dropped from the correlation calculation. `nan_threshold` should be between 0 and 1. Other options includes:

- `correlation_methods`: Methods to calculate the correlation. By default, only `pearson` correlation is calculated and shown. Can select one or more from `pearson`, `cramers_v`, and `correlation_ratio`. Or set to `all` to show all correlation charts.
- `correlation_target`: Defaults to `None`. It can be any columns of type `continuous`, `ordinal`, `categorical`

 BokehJS 1.4.0 successfully loaded.



 BokehJS 1.4.0 successfully loaded.



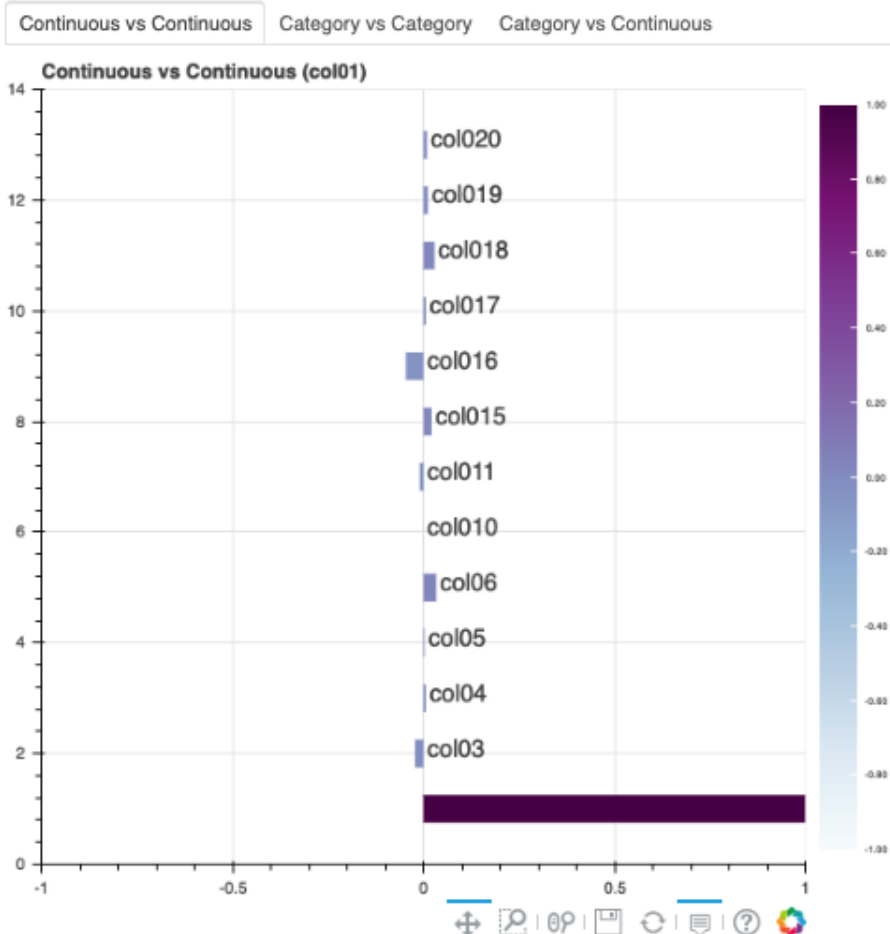
or zipcode. When `correlation_target` is set, only pairs that contain `correlation_target` display.

- `correlation_threshold`: Apply a filter to the correlation matrices and only exhibit the pairs whose correlation values are greater than or equal to the `correlation_threshold`.
- `force_recompute`: Defaults to False. Correlation matrices are cached. Set `force_recompute` to True to recalculate the correlation. Note that both `corr()` and `show_corr()` method can trigger calculation of correlation matrices if run with `force_recompute` set to be True, or when there is no cached value exists. `show_in_notebook()` calculates the correlation only when there are only numerical columns in the dataset.
- `frac`: Defaults to 1. The portion of the original data to calculate the correlation on. `frac` must be between 0 and 1.
- `plot_type`: Defaults to heatmap. Valid values are `heatmap` and `bar`. If `bar` is chosen, `correlation_target` also has to be set and the bar chart will only show the correlation values of the pairs which have the target in them.

```
ds.show_corr(correlation_target='col01', plot_type='bar')
```

BokehJS 1.4.0 successfully loaded.

The correlation matrix has been cached. Please make sure `overwrite=True` if you want to recalculate the correlation.



To explore features, use the `smart plot()` method. It accepts one or two feature names. The `show_in_notebook()`

method automatically determines the best type of plot based on the type of features that are to be plotted.

Three different examples are described. They use a binary classification dataset with 1,500 rows and 21 columns. 13 of the columns have a continuous data type, and 8 are categorical. There are three different examples.

- A single categorical feature: The `plot()` method detects that the feature is categorical because it only has the values of 0 and 1. It then automatically renders a plot of the count of each category.

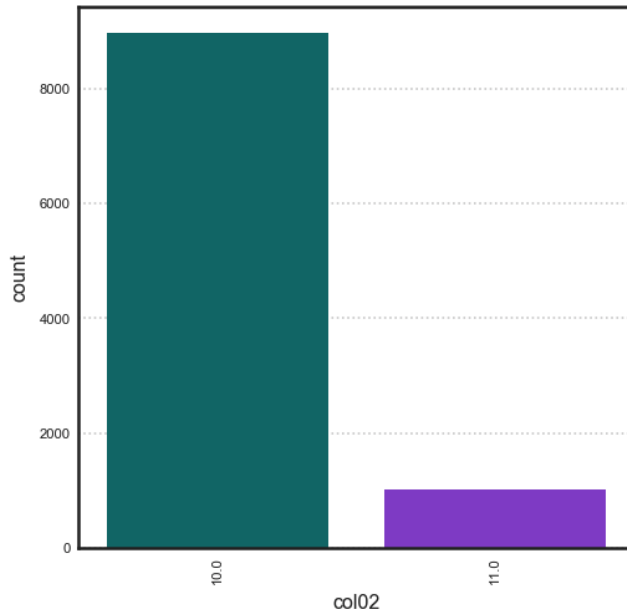
```
ds.plot("col02").show_in_notebook(figsize=(4,4))
```

## NOTE

Visualizations use a sampled dataset of size 10,000 (confidence level: 95, confidence interval: 1.0)

Set yscale to one of 'linear', 'log', 'symlog', 'logit' to apply scale to y axis

`_SINGLE_COLUMN_COUNT_PLOT, "col02" (categorical)`



- Categorical and continuous feature pair: ADS chooses the best plotting method, which is a violin plot.

```
ds.plot("col02", y="col01").show_in_notebook(figsize=(4,4))
```

- A pair of continuous features: ADS chooses a Gaussian heatmap as the best visualization. It generates a scatter plot and assigns a color to each data point based on the local density (Gaussian kernel).

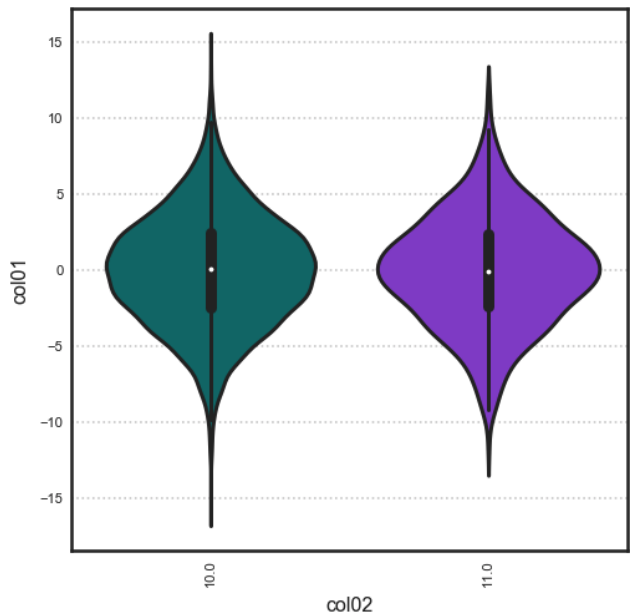
```
ds.plot("col01", y="col03").show_in_notebook()
```



**NOTE**

Visualizations use a sampled dataset of size 10,000 (confidence level: 95, confidence interval: 1.0)

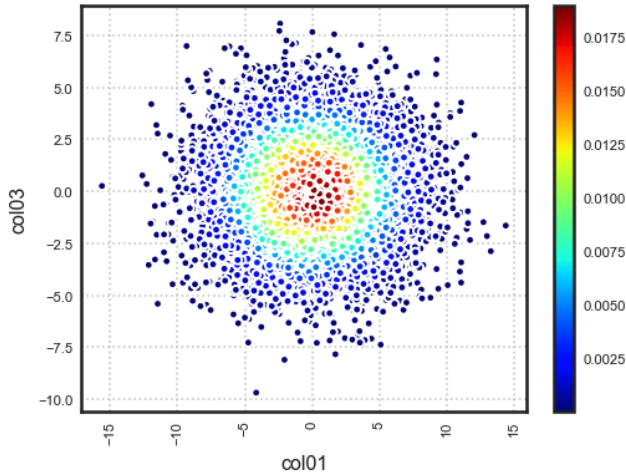
`_VIOLIN_PLOT`, "col02" (categorical) vs "col01" (continuous)



**NOTE**

Visualizations use a sampled dataset of size 10,000 (confidence level: 95, confidence interval: 1.0)

`_GAUSSIAN_HEATMAP`, "col01" (continuous) vs "col03" (continuous)



## 10.2 Customized

ADS provides intelligent default options for your plots. However, the visualization API is flexible enough to let you customize your charts or choose your own plotting library. You can use the `ADS call()` method to select your own plotting routine.

### 10.2.1 Seaborn

In this example, a dataframe is passed directly to the Seaborn pair plot function. It does a faceted, pairwise plot between all the features in the dataset. The function creates a grid of axes such that each variable in the data is shared in the y-axis across a row and in the x-axis across a column. The diagonal axes are treated differently by drawing a histogram of each feature.

```
import seaborn as sns
from sklearn.datasets import load_iris
import pandas as pd
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
sns.set(style="ticks", color_codes=True)
sns.pairplot(df.dropna())
```

### 10.2.2 Matplotlib

- Using Matplotlib:

```
import matplotlib.pyplot as plt
from numpy.random import randn

df = pd.DataFrame(randn(1000, 4), columns=list('ABCD'))

def ts_plot(df, figsize):
 ts = pd.Series(randn(1000), index=pd.date_range('1/1/2000', periods=1000))
 df.set_index(ts)
 df = df.cumsum()
 plt.figure()
 df.plot(figsize=figsize)
 plt.legend(loc='best')

ts_plot(df, figsize=(7,7))
```

- Using a Pie Chart:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = {'data': [1109, 696, 353, 192, 168, 86, 74, 65, 53]}
df = pd.DataFrame(data, index = ['20-50 km', '50-75 km', '10-20 km', '75-100 km',
 ↳ '3-5 km', '7-10 km', '5-7 km', '>100 km', '2-3 km'])

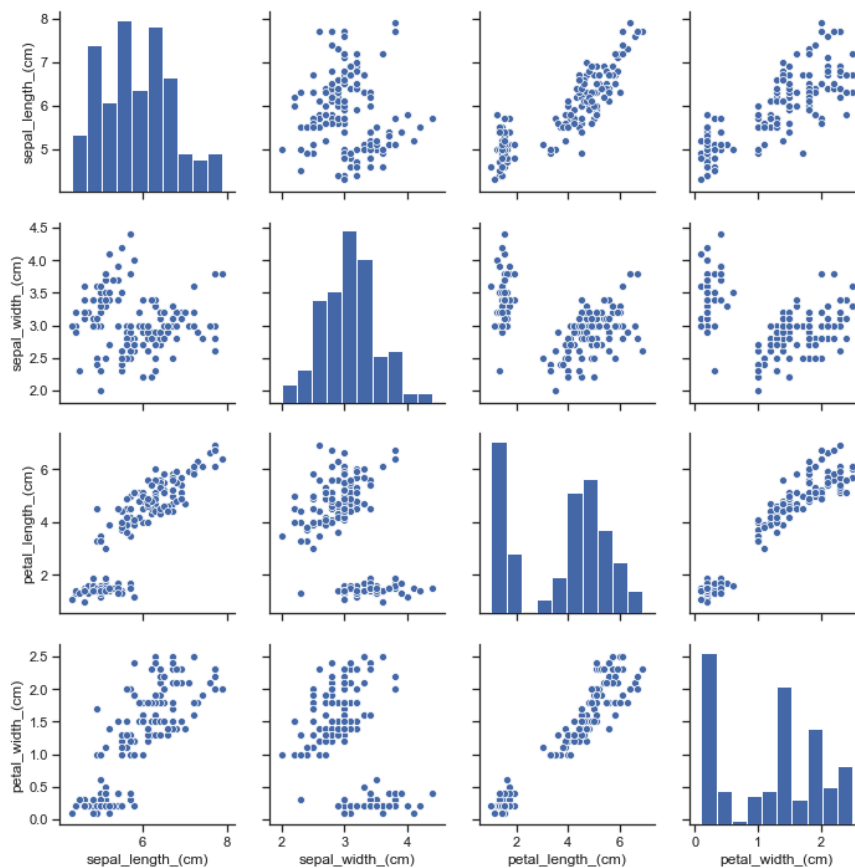
explode = (0, 0, 0, 0.1, 0.1, 0.2, 0.3, 0.4, 0.6)
```

(continues on next page)

Using entire dataset for graphing (150 rows)

Use `set_target()` to type the dataset for a particular learning task

<seaborn.axisgrid.PairGrid at 0x1153adc88>

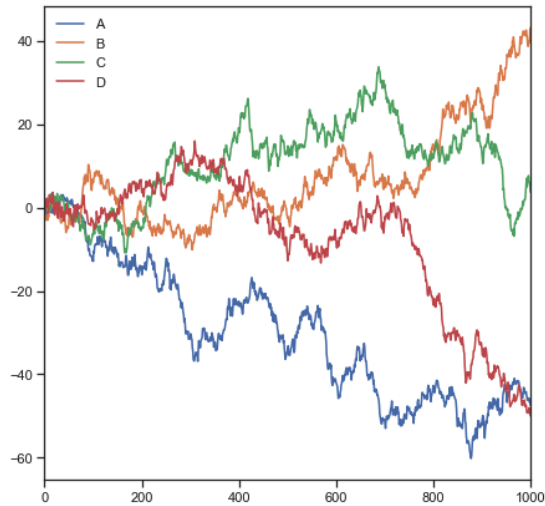


Using entire dataset for graphing (1000 rows)

**TIP:**

- + Use `show_in_notebook()` to visualize the dataset.
- + Use `get_recommendations()` to view and apply recommendations for dataset optimization.

<Figure size 432x288 with 0 Axes>



(continued from previous page)

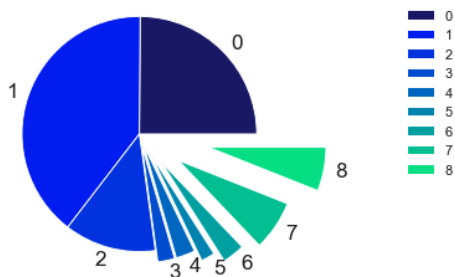
```
colors = ['#191970', '#001CF0', '#0038E2', '#0055D4', '#0071C6', '#008DB8', '#00AAAA',
 '#00C69C', '#00E28E', '#00FF80',]

def bar_plot(df, figsize):
 df["data"].plot(kind='pie', fontsize=17, colors=colors, explode=explode)
 plt.axis('equal')
 plt.ylabel('')
 plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
 plt.show()

bar_plot(df, figsize=(7,7))
```

Using entire dataset for graphing (9 rows)

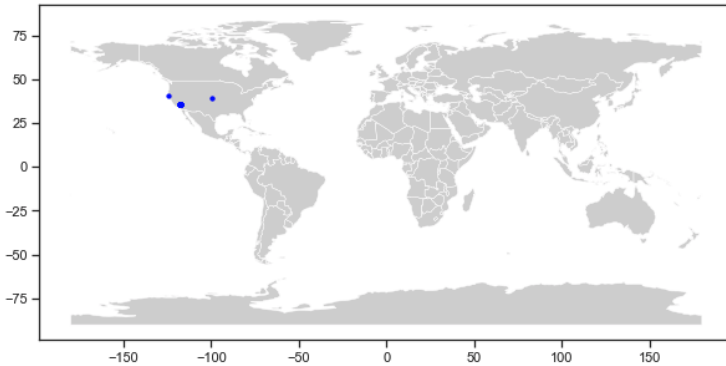
Use `set_target()` to type the dataset for a particular learning task



### 10.2.3 Geographic Information System (GIS)

This example uses the California earthquake data retrieved from United States Geological Survey (USGS) earthquake catalog. It visualizes the location of major earthquakes.

```
earthquake.plot_gis_scatter(lon="longitude", lat="latitude")
```





## TRAIN MODELS

In this section you will learn about model training on the Data Science cloud service using a variety of popular frameworks. This section covers the popular `sklearn` framework, along with gradient boosted tree estimators like `LightGBM` and `XGBoost`, Oracle AutoML and deep learning packages like `TensorFlow` and `PyTorch`.

The section covers how to serialize models and make use of the OCI Model Catalog to store model artifacts and meta data all using ADS to prepare the upload.

In the distributed training section you will see examples of how to work with `Dask`, `Horovod`, `TensorFlow` and `PyTorch` to do multinode training.

`TensorBoard` provides the visualization and the tooling that is needed to watch and record model training progress throughout the tuning stages.

### 11.1 ADSTuner

In addition to the other services for training models, ADS includes a hyperparameter tuning framework called `ADSTuner`.

`ADSTuner` supports using several hyperparameter search strategies that plug into common model architectures like `sklearn`.

`ADSTuner` further supports users defining their own search spaces and strategies. This makes `ADSTuner` functional and useful with any ML library that doesn't include hyperparameter tuning.

First, import the packages:

```
import category_encoders as ce
import lightgbm
import logging
import numpy as np
import os
import pandas as pd
import pytest
import sklearn
import xgboost

from ads.hpo.stopping_criterion import *
from ads.hpo.distributions import *
from ads.hpo.search_cv import ADSTuner, NotResumableError

from lightgbm import LGBMClassifier
from sklearn import preprocessing
```

(continues on next page)

(continued from previous page)

```

from sklearn.compose import ColumnTransformer
from sklearn.datasets import load_iris, load_boston
from sklearn.decomposition import PCA
from sklearn.ensemble import AdaBoostRegressor, AdaBoostClassifier
from sklearn.impute import SimpleImputer
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.metrics import make_scorer, f1_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from xgboost import XGBClassifier

```

This is an example of running the ADSTuner on a support model SGD from sklearn:

```

model = SGDClassifier() ##Initialize the model
X, y = load_iris(return_X_y=True)
X_train, X_valid, y_train, y_valid = train_test_split(X, y)
tuner = ADSTuner(model, cv=3) ## cv is cross validation splits
tuner.search_space() ##This is the default search space
tuner.tune(X_train, y_train, exit_criterion=[NTrials(10)])

```

ADSTuner generates a tuning report that lists its trials, best performing hyperparameters, and performance statistics with:

```

[I 2020-10-23 21:56:17,630] Trial 9 finished with value: 0.8316737790422001 and parameters: {'alpha': 0.0002576226059719444, 'penalty': 'l2'}. Best is trial 9 with value: 0.8316737790422001.
[I 2020-10-23 21:56:17,674] Trial 5 finished with value: 0.9106211474632527 and parameters: {'alpha': 0.07161796713234189, 'penalty': 'l2'}. Best is trial 5 with value: 0.9106211474632527.
[I 2020-10-23 21:56:17,792] Trial 3 finished with value: 0.9642010431484116 and parameters: {'alpha': 0.006158601374396708, 'penalty': 'none'}. Best is trial 3 with value: 0.9642010431484116.
[I 2020-10-23 21:56:17,891] Trial 4 finished with value: 0.7956377430061642 and parameters: {'alpha': 0.0008008011222908228, 'penalty': 'l2'}. Best is trial 3 with value: 0.9642010431484116.
[I 2020-10-23 21:56:17,903] Trial 6 finished with value: 0.9551920341394027 and parameters: {'alpha': 0.002629113116871369, 'penalty': 'l1'}. Best is trial 3 with value: 0.9642010431484116.
[I 2020-10-23 21:56:17,937] Trial 7 finished with value: 0.9642010431484116 and parameters: {'alpha': 0.0007283968106220585, 'penalty': 'none'}. Best is trial 3 with value: 0.9642010431484116.
[I 2020-10-23 21:56:17,940] Trial 1 finished with value: 0.9551920341394026 and parameters: {'alpha': 0.0003638169088886491, 'penalty': 'l1'}. Best is trial 3 with value: 0.9642010431484116.
[I 2020-10-23 21:56:17,955] Trial 2 pruned. trial was pruned at iteration 99.
[I 2020-10-23 21:56:18,097] Trial 8 finished with value: 0.9732100521574205 and parameters: {'alpha': 0.006335356664818435, 'penalty': 'l1'}. Best is trial 8 with value: 0.9732100521574205.
[I 2020-10-23 21:56:18,101] Trial 0 finished with value: 0.9642010431484116 and parameters: {'alpha': 0.0013210136796797667, 'penalty': 'l1'}. Best is trial 8 with value: 0.9732100521574205.
CPU times: user 16.4 s, sys: 8.99 s, total: 25.3 s
Wall time: 16.4 s

```

You can use `tuner.best_score` to get the best score on the scoring metric used (accessible as `tuner.scoring_name`). The best selected parameters are obtained with `tuner.best_params` and the complete record of trials with `tuner.trials`.

If you have further compute resources and want to continue hyperparameter optimization on a model that has already been optimized, you can use:

```

tuner.resume(exit_criterion=[TimeBudget(5)], loglevel=logging.NOTSET)
print('So far the best {} score is {}'.format(tuner.scoring_name, tuner.best_score))
print("The best trial found was number: " + str(tuner.best_index))

```

ADSTuner has some robust visualization and plotting capabilities:

```

tuner.plot_best_scores()
tuner.plot_intermediate_scores()
tuner.search_space()
tuner.plot_contour_scores(params=['penalty', 'alpha'])

```

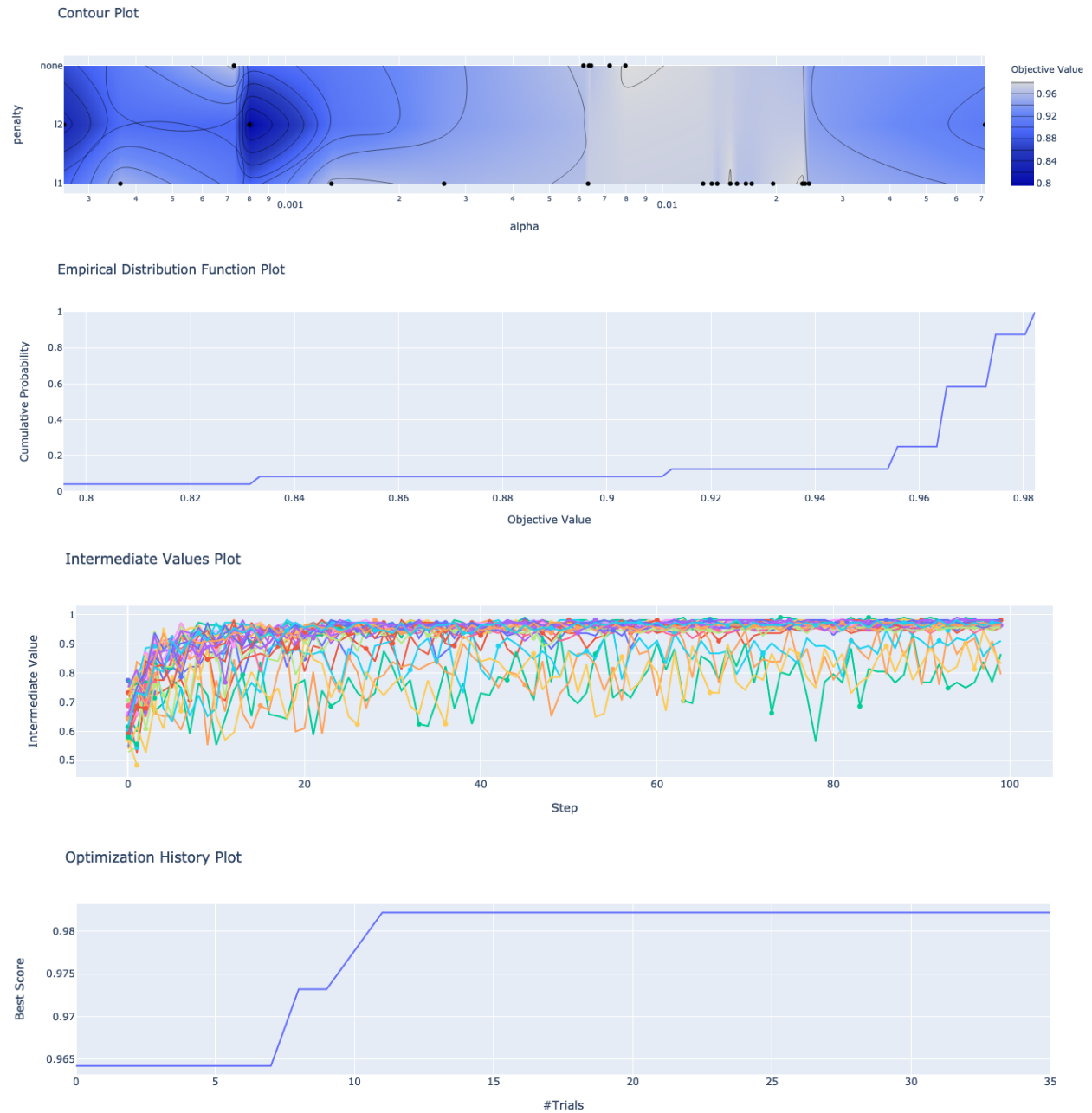
(continues on next page)

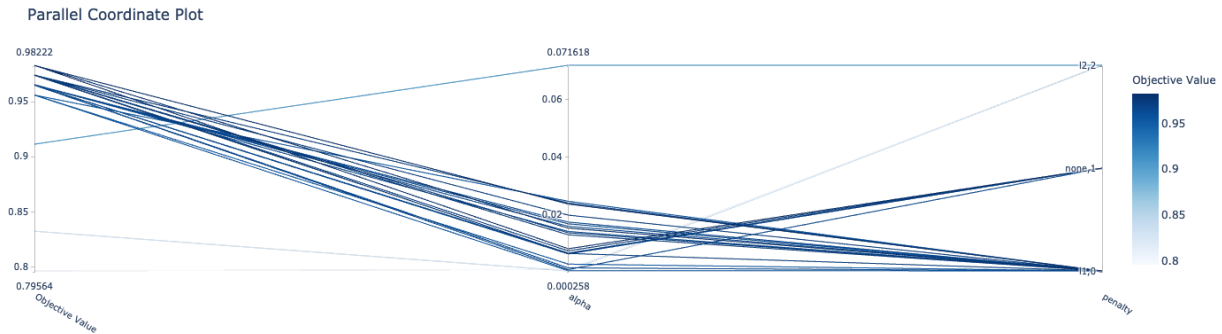


(continued from previous page)

```
tuner.plot_parallel_coordinate_scores(params=['penalty', 'alpha'])
tuner.plot_edf_scores()
```

These commands produce the following plots:





ADSTuner supports custom scoring functions and custom search spaces. This example uses a different model:

```
model2 = LogisticRegression()
tuner = ADSTuner(model2,
 strategy = {
 'C': LogUniformDistribution(low=1e-05, high=1),
 'solver': CategoricalDistribution(['saga']),
 'max_iter': IntUniformDistribution(500, 1000, 50)},
 scoring=make_scorer(f1_score, average='weighted'),
 cv=3)
tuner.tune(X_train, y_train, exit_criterion=[NTrials(5)])
```

ADSTuner doesn't support every model. The supported models are:

- 'Ridge',
- 'RidgeClassifier',
- 'Lasso',
- 'ElasticNet',
- 'LogisticRegression',
- 'SVC',
- 'SVR',
- 'LinearSVC',
- 'LinearSVR',
- 'DecisionTreeClassifier',
- 'DecisionTreeRegressor',
- 'RandomForestClassifier',
- 'RandomForestRegressor',
- 'GradientBoostingClassifier',
- 'GradientBoostingRegressor',
- 'XGBClassifier',
- 'XGBRegressor',
- 'ExtraTreesClassifier',
- 'ExtraTreesRegressor',
- 'LGBMClassifier',

- 'LGBMRegressor',
- 'SGDClassifier',
- 'SGDRegressor'

The AdaBoostRegressor model is not supported. This is an example of a custom strategy to use with this model:

```
model3 = AdaBoostRegressor()
X, y = load_boston(return_X_y=True)
X_train, X_valid, y_train, y_valid = train_test_split(X, y)
tuner = ADSTuner(model3, strategy={'n_estimators': IntUniformDistribution(50, 100)})
tuner.tune(X_train, y_train, exit_criterion=[TimeBudget(5)])
```

Finally, ADSTuner supports sklearn pipelines:

```
df, target = pd.read_csv(os.path.join('~', 'advanced-ds', 'tests', 'vor_datasets', 'vor_
↳ titanic.csv')), 'Survived'
X = df.drop(target, axis=1)
y = df[target]

numeric_features = X.select_dtypes(include=['int64', 'float64', 'int32', 'float32']).
↳ columns
categorical_features = X.select_dtypes(include=['object', 'category', 'bool']).columns

y = preprocessing.LabelEncoder().fit_transform(y)

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.3, random_
↳ state=42)

num_features = len(numeric_features) + len(categorical_features)

numeric_transformer = Pipeline(steps=[
 ('num_imputer', SimpleImputer(strategy='median')),
 ('num_scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
 ('cat_imputer', SimpleImputer(strategy='constant', fill_value='missing')),
 ('cat_encoder', ce.woe.WOEEncoder())
])

preprocessor = ColumnTransformer(
 transformers=[
 ('num', numeric_transformer, numeric_features),
 ('cat', categorical_transformer, categorical_features)
]
)

pipe = Pipeline(
 steps=[
 ('preprocessor', preprocessor),
 ('feature_selection', SelectKBest(f_classif, k=int(0.9 * num_features))),
 ('classifier', LogisticRegression())
]
)
```

(continues on next page)

(continued from previous page)

```
)

def customize_score(y_true, y_pred, sample_weight=None):
 score = y_true == y_pred
 return np.average(score, weights=sample_weight)

score = make_scorer(customize_score)
ads_search = ADSTuner(
 pipe,
 scoring=score,
 strategy='detailed',
 cv=2,
 random_state=42
)
ads_search.tune(X=X_train, y=y_train, exit_criterion=[NTrials(20)])
```

### 11.1.1 Notebook Example: Hyperparameter Optimization with ADSTuner

#### Overview:

A hyperparameter is a parameter that is used to control a learning process. This is in contrast to other parameters that are learned in the training process. The process of hyperparameter optimization is to search for hyperparameter values by building many models and assessing their quality. This notebook provides an overview of the ADSTuner hyperparameter optimization engine. ADSTuner can optimize any estimator object that follows the [scikit-learn API](#).

#### Objectives:

- Introduction
  - Synchronous Tuning with Exit Criterion Based on Number of Trials
  - Asynchronously Tuning with Exit Criterion Based on Time Budget
  - Inspecting the Tuning Trials
- Defining a Custom Search Space and Score
  - Changing the Search Space Strategy
- Optimizing a scikit-learn Pipeline()
- References

---

#### Important:

Placeholder text for required values are surrounded by angle brackets that must be removed when adding the indicated content. For example, when adding a database name to `database_name = "<database_name>"` would become `database_name = "production"`.

---

Datasets are provided as a convenience. Datasets are considered third party content and are not considered materials under your agreement with Oracle applicable to the services. The iris dataset is distributed under the [BSD license](#).

```

import category_encoders as ce
import lightgbm
import logging
import numpy as np
import os
import pandas as pd
import sklearn
import time

from ads.hpo.stopping_criterion import *
from ads.hpo.distributions import *
from ads.hpo.search_cv import ADSTuner, State

from sklearn import preprocessing
from sklearn.compose import ColumnTransformer
from sklearn.datasets import load_iris, load_boston
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.metrics import make_scorer, f1_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif

```

## Introduction

Hyperparameter optimization requires a model, dataset, and an ADSTuner object to perform the search.

ADSTuner() Performs a hyperparameter search using [cross-validation](#). You can specify the number of folds you want to use with the cv parameter.

Because the ADSTuner() needs a search space in which to tune the hyperparameters, you must use the `strategy` parameter. This parameter can be set in two ways. You can specify detailed search criteria or you can use the built-in defaults. For the supported model classes, ADSTuner provides `perfunctory` and `detailed` search spaces that are optimized for the chosen class of model. The `perfunctory` option is optimized for a small search space so that the most important hyperparameters are tuned. Generally, this option is used early in your search as it reduces the computational cost and allows you to assess the quality of the model class that you are using. The `detailed` search space instructs ADSTuner to cover a broad search space by tuning more hyperparameters. Typically, you would use it when you have determined what class of model is best suited for the dataset and type of problem you are working on. If you have experience with the dataset and have a good idea of what the best hyperparameter values are, you can explicitly specify the search space. You pass a dictionary that defines the search space into the `strategy`.

The parameter `storage` takes a database URL. For example, `sqlite:///home/datascience/example.db`. When `storage` is set to the default value `None`, a new `sqlite` database file is created internally in the `tmp` folder with a unique name. The name format is `sqlite:///tmp/hpo_*.db`. `study_name` is the name of this study for this ADSTuner object. Each ADSTuner object has a unique `study_name`. However, one database file can be shared among different ADSTuner objects. `load_if_exists` controls whether to load an existing study from an existing database file. If `False`, it raises a `DuplicatedStudyError` when the `study_name` exists.

The `loglevel` parameter controls the amount of logging information displayed in the notebook.

This notebook uses the `scikit-learn` `SGDClassifier()` model and the `iris` dataset. This model object is a regularized linear model with [stochastic gradient descent](#) (SGD) used to optimize the model parameters.

The next cell creates the `SGDClassifier()` model, initialize an ADSTuner object, and loads the `iris` data.

```
tuner = ADSTuner(SGDClassifier(), cv=3, loglevel=logging.WARNING)
X, y = load_iris(return_X_y=True)
```

```
[32m[I 2021-04-21 20:04:03,435][0m A new study created with name: hpo_22cfd4d5-c512-4e84-
↪b7f8-d6d9c721ff05[0m
```

Each model class has a set of hyperparameters that you need to optimized. The `strategy` attribute returns what strategy is being used. This can be `perfunctory`, `detailed`, or a dictionary that defines the strategy. The method `search_space()` always returns a dictionary of hyperparameters that are to be searched. Any hyperparameter that is required by the model, but is not listed, uses the default value that is defined by the model class. To see what search space is being used for your model class when `strategy` is `perfunctory` or `detailed` use the `search_space()` method to see the details.

The `adstuner_search_space_update.ipynb` notebook has detailed examples about how to work with and update the search space.

The next cell displaces the search strategy and the search space.

```
print(f'Search Space for strategy "{tuner.strategy}" is: \n {tuner.search_space()}')
```

```
Search Space for strategy "perfunctory" is:
{'alpha': LogUniformDistribution(low=0.0001, high=0.1), 'penalty':
↪CategoricalDistribution(choices=['l1', 'l2', 'none'])}
```

The `tune()` method starts a tuning process. It has a synchronous and asynchronous mode for tuning. The mode is set with the `synchronous` parameter. When it is set to `False`, the tuning process runs asynchronously so it runs in the background and allows you to continue your work in the notebook. When `synchronous` is set to `True`, the notebook is blocked until `tune()` finishes running. The `adntuner_sync_and_async.ipynb` notebook illustrates this feature in a more detailed way.

The `ADSTuner` object needs to know when to stop tuning. The `exit_criterion` parameter accepts a list of criteria that cause the tuning to finish. If any of the criteria are met, then the tuning process stops. Valid exit criteria are:

- `NTrials(n)`: Run for `n` number of trials.
- `TimeBudget(t)`: Run for `t` seconds.
- `ScoreValue(s)`: Run until the score value exceeds `s`.

The default behavior is to run for 50 trials (`NTrials(50)`).

The stopping criteria are listed in the `ads.hpo.stopping_criterion` module.

### Synchronous Tuning with Exit Criterion Based on Number of Trials

This section demonstrates how to perform a synchronous tuning process with the exit criteria based on the number of trials. In the next cell, the `synchronous` parameter is set to `True` and the `exit_criterion` is set to `[NTrials(5)]`.

```
tuner.tune(X, y, exit_criterion=[NTrials(5)], synchronous=True)
```

You can access a summary of the trials by looking at the various attributes of the `tuner` object. The `scoring_name` attribute is a string that defines the name of the scoring metric. The `best_score` attribute gives the best score of all the completed trials. The `best_params` parameter defines the values of the hyperparameters that have to lead to the best score. Hyperparameters that are not in the search criteria are not reported.

```
print(f"So far the best {tuner.scoring_name} score is {tuner.best_score} and the best
↪hyperparameters are {tuner.best_params}")
```

```
So far the best mean accuracy score is 0.966666666666667 and the best hyperparameters_
are {'alpha': 0.002623793623610696, 'penalty': 'none'}
```

You can also look at the detailed table of all the trials attempted:

```
tuner.trials.tail()
```

### Asynchronously Tuning with Exit Criterion Based on Time Budget

ADSTuner() tuner can be run in an asynchronous mode by setting `synchronous=False` in the `tune()` method. This allows you to run other Python commands while the tuning process is executing in the background. This section demonstrates how to run an asynchronous search for the optimal hyperparameters. It uses a stopping criteria of five seconds. This is controlled by the parameter `exit_criterion=[TimeBudget(5)]`.

The next cell starts an asynchronous tuning process. A loop is created that prints the best search results that have been detected so far by using the `best_score` attribute. It also displays the remaining time in the time budget by using the `time_remaining` attribute. The attribute `status` is used to exit the loop.

```
This cell will return right away since it's running asynchronously.
tuner.tune(exit_criterion=[TimeBudget(5)])
while tuner.status == State.RUNNING:
 print(f"So far the best score is {tuner.best_score} and the time left is {tuner.time_
remaining}")
 time.sleep(1)
```

```
So far the best score is 0.966666666666667 and the time left is 4.977275848388672
So far the best score is 0.966666666666667 and the time left is 3.9661824703216553
So far the best score is 0.966666666666667 and the time left is 2.9267797470092773
So far the best score is 0.966666666666667 and the time left is 1.912914752960205
So far the best score is 0.973333333333333 and the time left is 0.9021461009979248
So far the best score is 0.973333333333333 and the time left is 0
```

The attribute `best_index` give you the index in the `trials` data frame where the best model is located.

```
tuner.trials.loc[tuner.best_index, :]
```

```
number 10
value 0.98
datetime_start 2021-04-21 20:04:17.013347
datetime_complete 2021-04-21 20:04:18.623813
duration 0 days 00:00:01.610466
params_alpha 0.014094
params_penalty 11
user_attrs_mean_fit_time 0.16474
user_attrs_mean_score_time 0.024773
user_attrs_mean_test_score 0.98
user_attrs_metric mean accuracy
user_attrs_split0_test_score 1.0
user_attrs_split1_test_score 1.0
user_attrs_split2_test_score 0.94
user_attrs_std_fit_time 0.006884
user_attrs_std_score_time 0.00124
user_attrs_std_test_score 0.028284
```

(continues on next page)

(continued from previous page)

```
state COMPLETE
Name: 10, dtype: object
```

The attribute `n_trials` reports the number of successfully completed trials.

```
print(f"The total of trials was: {tuner.n_trials}.")
```

```
The total of trials was: 11.
```

### Inspecting the Tuning Trials

You can inspect the tuning trials performance using several built-in plots.

**Note:** If the tuning process is still running in the background, the plot runs in real time to update the new changes until the tuning process completes.

```
tuner.tune(exit_criterion=[NTrials(5)], loglevel=logging.WARNING) # uncomment this
↪ line to see the real-time plot.
tuner.plot_best_scores()
```

```
tuner.plot_intermediate_scores()
```

```
tuner.plot_contour_scores(params=['penalty', 'alpha'])
```

```
tuner.plot_parallel_coordinate_scores(params=['penalty', 'alpha'])
```

```
tuner.plot_edf_scores()
```

```
tuner.plot_param_importance()
```

Waiting **for** more trials before evaluating the param importance.

### Defining a Custom Search Space and Score

Instead of using a `perfunctory` or detailed strategy, define a custom search space strategy.

The next cell, creates a `LogisticRegression()` model instance then defines a custom search space strategy for the three `LogisticRegression()` hyperparameters, `C`, `solver`, and `max_iter` parameters.

You can define a custom scoring parameter, see `Optimizing a scikit-learn Pipeline()` though this example uses the standard weighted average  $F_1$ , `f1_score`.

```
tuner = ADSTuner(LogisticRegression(),
 strategy = {'C': LogUniformDistribution(low=1e-05, high=1),
 'solver': CategoricalDistribution(['saga']),
 'max_iter': IntUniformDistribution(500, 2000, 50)},
 scoring=make_scorer(f1_score, average='weighted'),
 cv=3)
tuner.tune(X, y, exit_criterion=[NTrials(5)], synchronous=True, loglevel=logging.WARNING)
```

### Changing the Search Space Strategy

You can change the search space in the following three ways:

- Add new hyperparameters



- Remove existing hyperparameters
- Modify the range of existing non-categorical hyperparameters

**Note:** You can't change the distribution of an existing hyperparameter or make any changes to a hyperparameter that is based on a categorical distribution. You need to initiate a new ADSTuner object for those cases. For more detailed information, review the `adstuner_search_space_update.ipynb` notebook.

The next cell switches to a detailed strategy. All previous values set for `C`, `solver`, and `max_iter` are kept, and ADSTuner infers distributions for the remaining hyperparameters. You can force an overwrite by setting `overwrite=True`.

```
tuner.search_space(strategy='detailed')
```

```
{'C': LogUniformDistribution(low=1e-05, high=10),
'solver': CategoricalDistribution(choices=['saga']),
'max_iter': IntUniformDistribution(low=500, high=2000, step=50),
'dual': CategoricalDistribution(choices=[False]),
'penalty': CategoricalDistribution(choices=['elasticnet']),
'11_ratio': UniformDistribution(low=0, high=1)}
```

Alternatively, you can edit a subset of the search space by changing the range.

```
tuner.search_space(strategy={'C': LogUniformDistribution(low=1e-05, high=1)})
```

```
{'C': LogUniformDistribution(low=1e-05, high=1),
'solver': CategoricalDistribution(choices=['saga']),
'max_iter': IntUniformDistribution(low=500, high=2000, step=50),
'dual': CategoricalDistribution(choices=[False]),
'penalty': CategoricalDistribution(choices=['elasticnet']),
'11_ratio': UniformDistribution(low=0, high=1)}
```

Here's an example of using `overwrite=True` to reset to the default values for detailed:

```
tuner.search_space(strategy='detailed', overwrite=True)
```

```
{'C': LogUniformDistribution(low=1e-05, high=10),
'dual': CategoricalDistribution(choices=[False]),
'penalty': CategoricalDistribution(choices=['elasticnet']),
'solver': CategoricalDistribution(choices=['saga']),
'11_ratio': UniformDistribution(low=0, high=1)}
```

```
tuner.tune(X, y, exit_criterion=[NTrials(5)], synchronous=True, loglevel=logging.WARNING)
```

### Optimizing a scikit-learn Pipeline

The following example demonstrates how the ADSTuner hyperparameter optimization engine can optimize the `sklearn Pipeline()` objects.

You create a scikit-learn `Pipeline()` model object and use ADSTuner to optimize its performance on the iris dataset from sklearn.

The dataset is then split into `X` and `y`, which refers to the training features and the target feature respectively. Again, applying a `train_test_split()` call splits the data into training and validation datasets.

```

X, y = load_iris(return_X_y=True)
X = pd.DataFrame(data=X, columns=["sepal_length", "sepal_width", "petal_length", "petal_
↪width"])
y = pd.DataFrame(data=y)

numeric_features = X.select_dtypes(include=['int64', 'float64', 'int32', 'float32']).
↪columns
categorical_features = y.select_dtypes(include=['object', 'category', 'bool']).columns

y = preprocessing.LabelEncoder().fit_transform(y)

num_features = len(numeric_features) + len(categorical_features)

numeric_transformer = Pipeline(steps=[
 ('num_imputer', SimpleImputer(strategy='median')),
 ('num_scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
 ('cat_imputer', SimpleImputer(strategy='constant', fill_value='missing')),
 ('cat_encoder', ce.woe.WOEEncoder())
])

preprocessor = ColumnTransformer(
 transformers=[
 ('num', numeric_transformer, numeric_features),
 ('cat', categorical_transformer, categorical_features)
]
)

pipe = Pipeline(
 steps=[
 ('preprocessor', preprocessor),
 ('feature_selection', SelectKBest(f_classif, k=int(0.9 * num_features))),
 ('classifier', LogisticRegression())
]
)

```

You can define a custom score function. In this example, it is directly measuring how close the predicted y-values are to the true y-values by taking the weighted average of the number of direct matches between the y-values.

```

def custom_score(y_true, y_pred, sample_weight=None):
 score = (y_true == y_pred)
 return np.average(score, weights=sample_weight)

score = make_scorer(custom_score)

```

Again, you instantiate the `ADSTuner()` object and use it to tune the iris` dataset:

```

ads_search = ADSTuner(
 pipe,
 scoring=score,
 strategy='detailed',

```

(continues on next page)

(continued from previous page)

```

cv=2,
random_state=42)

ads_search.tune(X=X, y=y, exit_criterion=[NTrials(20)], synchronous=True,
↳ loglevel=logging.WARNING)

```

The `ads_search` tuner can provide useful information about the tuning process, like the best parameter that was optimized, the best score achieved, the number of trials, and so on.

```
ads_search.sklearn_steps
```

```
{'classifier__C': 9.47220908749299,
 'classifier__dual': False,
 'classifier__l1_ratio': 0.9967712201895031,
 'classifier__penalty': 'elasticnet',
 'classifier__solver': 'saga'}
```

```
ads_search.best_params
```

```
{'C': 9.47220908749299,
 'dual': False,
 'l1_ratio': 0.9967712201895031,
 'penalty': 'elasticnet',
 'solver': 'saga'}
```

```
ads_search.best_score
```

```
0.9733333333333334
```

```
ads_search.best_index
```

```
12
```

```
ads_search.trials.head()
```

```
ads_search.n_trials
```

```
20
```

## References

- [ADS Library Documentation](#)
- [Cross-Validation](#)
- [OCI Data Science Documentation](#)
- [Oracle Data & AI Blog](#)
- [Stochastic Gradient Descent](#)

## 11.2 Distributed Training

---

### Distributed Training with OCI Data Science

This documentation shows you how to preprocess, and train on a machine learning model, using Oracle Cloud Infrastructure. This section will not teach you about distributed training, instead it will help you run your existing distributed training code on OCI Data Science.

---

Distributed training is the process of taking a training workload which comprises training code and training data and making both of these available in a cluster.

The conceptual difference with distributed training is that multiple workers coordinated in a cluster running on multiple VM instances allows horizontal scaling of parallelizable tasks. While single node training is well suited to traditional ML models, very large datasets or compute intensive workloads like deep learning and deep neural networks, tends to be better suited to distributed computing environments.

Distributed Training benefits two classes of problem, one where the data is parallelizable, the other where the model network is parallelizable. The most common and easiest to develop is data parallelism. Both forms of parallelism can be combined to handle both large models and large datasets.

#### Data Parallelism

In this form of distributed training the training data is partitioned into some multiple of the number of nodes in the compute cluster. Each node holds the model and is in communication with other node participating in a coordinated optimization effort.

Sometimes data sampling is possible, but often at the expense of model accuracy. With distributed training you can avoid having to sample the data to fit a single node.

#### Model Parallelism

This form of distributed training is used when workers need to worker nodes need to synchronize and share parameters. The data fits into the memory of each worker, but the training takes too long. With model parallelism more epochs can run and more hyper-parameters can be explored.

### Distributed Training with OCI Data Science

To outline the process by which you create distributed training workloads is the same regardless of framework used. Sections of the configuration differ between frameworks but the experience is consistent. The user brings only the (framework specific) training python code, along with the yaml declarative definition.

ADS makes use of yaml to express configurations. The yaml specification has sections to describe the cluster infrastructure, the python runtime code, and the cluster framework.

The architecture is extensible to support well known frameworks and future versions of these. The set of service provided frameworks for distributed training include:

- [Dask](#) for LightGBM, XGBoost, Scikit-Learn, and Dask-ML
- [Horovod](#) for PyTorch & Tensorflow
- [PyTorch Distributed](#) for PyTorch native using `DistributedDataParallel` - no training code changes to run PyTorch model training on a cluster. You can use Horovod to do the same, which has some advanced features like auto-tuning to improve allreduce performance, and fp16 gradient compression.
- [Tensorflow Distributed](#) for Tensorflow distributed training strategies like `MirroredStrategy`, `MultiWorkerMirroredStrategy` and `ParameterServerStrategy`

## 11.2.1 Getting Started

### 11.2.1.1 Key Steps

1. Initialize the code workspace to prepare for distributed training
2. Build container image
3. Tag and Push the image to ocir
4. Define the cluster requirement using *YAML Spec*
5. Start distributed training using `ads opctl run -f <yaml file>`
6. Monitor the job using `ads jobs watch <main job run id>`

### 11.2.1.2 Prepare container image for distributed workload

**Prerequisite:**

1. Internet Connection
2. ADS cli is *installed*
3. Docker engine

```
ads opctl distributed-training init --framework <framework choice>
```

To run a distributed workload on OCI Data Science Jobs, you need prepare a container image with the source code that you want to run and the framework (Dask|Horovod|PyTorch) setup. OCI Data Science provides you with the Dockerfiles and bootstrapping scripts to build framework specific container images. This step creates a folder in the current working directory called `oci_distributed_training`. This folder contains all the artifacts required to setup and bootstrap the framework code. Refer to `README.md` file to see more details on how to build and push the container image to the ocir

### 11.2.1.3 Check Config File generated by the Main and the Worker Nodes

**Prerequisite:**

1. A cluster that is in In-Progress, Succeeded or Failed (Supported only in some cases)
2. Job OCID and the `work_dir` of the cluster or a yaml file which contains all the details displayed during cluster creation.

```
ads opctl distributed-training show-config -f <cluster yaml file>
```

The main node generates `MAIN_config.json` and worker nodes generate `WORKER_<job_run_ocid>_config.json`. You may want to check the configuration details for find the IP address of the Main node. This can be useful to bring up dashboard for dask or debugging.

## 11.2.2 Configurations

### 11.2.2.1 Networks

You need to use a private subnet for distributed training and configure the security list to allow traffic through specific ports for communication between nodes. The following default ports are used by the corresponding frameworks:

- *Dask*:
  - Scheduler Port: **8786**. More information [here](#)
  - Dashboard Port: **8787**. More information [here](#)
  - Worker Ports: **Default is Random**. It is good to open a specific range of port and then provide the value in the startup option. More information [here](#)
  - Nanny Process Ports: **Default is Random**. It is good to open a specific range of port and then provide the value in the startup option. More information [here](#)
- *PyTorch*: By default, PyTorch uses **29400**.
- *Horovod*: allow TCP traffic on all ports within the subnet.
- *Tensorflow*: Worker Port: Allow traffic from all source ports to one worker port (default: 12345). If changed, provide this in train.yaml config.

See also: [Security Lists](#)

### 11.2.2.2 OCI Policies

Several OCI policies are needed for distributed training.

---

#### Policy subject

In the following example, group `<your_data_science_users>` is the subject of the policy. When starting the job from an OCI notebook session using resource principal, the subject should be `dynamic-group`, for example, `dynamic-group <your_notebook_sessions>`

---

Distributed training uses [OCI Container Registry](#) to store the container image.

To push images to container registry, the `manage repos` policy is needed, for example:

```
Allow group <your_data_science_users> to manage repos in compartment <your_compartment_
↪name>
```

To pull images from container registry for local testing, the `use repos` policy is needed, for example:

```
Allow group <your_data_science_users> to read repos in compartment <your_compartment_
↪name>
```

You can also restrict the permission to specific repository, for example:

```
Allow group <your_data_science_users> to read repos in compartment <your_compartment_
↪name> where all { target.repo.name=<your_repo_name> }
```

See also: [Policies to Control Repository Access](#)

To start distributed training jobs, the user will need access to multiple resources, including:

- read repos
- manage data-science-jobs
- manage data-science-job-runs
- use virtual-network-family
- manage log-groups
- use log-content
- read metrics

For example:

```
Allow group <your_data_science_users> to manage data-science-jobs in compartment <your_
↪compartment_name>
Allow group <your_data_science_users> to manage data-science-job-runs in compartment
↪<your_compartment_name>
Allow group <your_data_science_users> to use virtual-network-family in compartment <your_
↪compartment_name>
Allow group <your_data_science_users> to manage log-groups in compartment <your_
↪compartment_name>
Allow group <your_data_science_users> to use logging-family in compartment <your_
↪compartment_name>
Allow group <your_data_science_users> to use read metrics in compartment <your_
↪compartment_name>
```

We also need policies for job runs, for example:

```
Allow dynamic-group <distributed_training_job_runs> to read repos in compartment <your_
↪compartment_name>
Allow dynamic-group <distributed_training_job_runs> to use data-science-family in
↪compartment <your_compartment_name>
Allow dynamic-group <distributed_training_job_runs> to use virtual-network-family in
↪compartment <your_compartment_name>
Allow dynamic-group <distributed_training_job_runs> to use log-groups in compartment
↪<your_compartment_name>
Allow dynamic-group <distributed_training_job_runs> to use logging-family in compartment
↪<your_compartment_name>
```

See also [Data Science Policies](#).

Distributed training uses OCI Object Storage to store artifacts and outputs. The bucket should be created before starting any distributed training. The `manage objects` policy is needed for users and job runs to read/write files in the bucket. The `manage buckets` policy is required for job runs to synchronize generated artifacts. For example:

```
Allow group <your_data_science_users> to manage objects in compartment your_compartment_
↪name where all {target.bucket.name=<your_bucket_name>}
Allow dynamic-group <distributed_training_job_runs> to manage objects in compartment
↪your_compartment_name where all {target.bucket.name=<your_bucket_name>}
Allow dynamic-group <distributed_training_job_runs> to manage buckets in compartment
↪your_compartment_name where all {target.bucket.name=<your_bucket_name>}
```

See also [Object Storage Policies](#)

### 11.2.2.3 Policy Syntax

The overall syntax of a policy statement is as follows:

Allow <subject> to <verb> <resource-type> in <location> where <conditions>

See also: <https://docs.oracle.com/en-us/iaas/Content/Identity/Concepts/policysyntax.htm>

For <subject>:

- If you are using API key authentication, <subject> should be the group your user belongs to. For example, group <your\_data\_science\_users>.
- If you are using resource principal or instance principal authentication, <subject> should be the dynamic group to which your OCI resource belongs. Here the resource is where you initialize the API requests, which is usually a job run, a notebook session or compute instance. For example, dynamic-group <distributed\_training\_job\_runs>

**Dynamic group** allows you to group OCI resources like job runs and notebook sessions. Distributed training is running on Data Science Jobs, for the training process to access resources, the job runs need to be defined as a dynamic group and use as the <subject> for policies.

In the following examples, we define distributed\_training\_job\_runs dynamic group as:

```
all { resource.type='datasciencejobrun', resource.compartment.id='<job_run_compartment_ocid>'
}
```

We also assume the user in group <your\_data\_science\_users> is preparing the docker image and starting the training job.

The <verb> determines the ability of the <subject> to work on the <resource-type>. Four options are available: inspect, read, user and manage.

The <resource-type> specifies the resources we would like to access. Distributed training uses the following OCI resources/services:

- **Data Science Jobs**. Resource Type: data-science-jobs and data-science-job-runs
- **Object Storage**. Resource Type: buckets and objects
- **Container Registry**. Resource Type: repos

The <location> is usually the compartment or tenancy that your resources (specified by <resource-type>) resides.

\* If you would like the <subject> to have access to all resources (specified by <resource-type>) in the tenancy, you can use tenancy as <location>. \* If you would like the <subject> to have access to resources in specific compartment, you can use compartment your\_compartment\_name as <location>.

The where <conditions> can be used to filter the resources specified in <resource-type>.

## 11.2.3 Developer Guide

### 11.2.3.1 Build Image

---

#### Tip

Use -h option to see options and usage help

```
ads opctl distributed-training build-image -h
```

---

#### Args



- -t: Tag of the docker image
- -reg: Docker Repository
- -df: Dockerfile using which docker will be build
- -push: push the image to oci registry
- -s: source code dir

```
ads opctl distributed-training build-image \
-t $TAG \
-reg $NAME_OF_REGISTRY \
-df $PATH_TO_DOCKERFILE \
-s $MOUNT_FOLDER_PATH
```

**Note :**

This command can be used to build a docker image from ads CLI. It writes the config.ini file in the user's runtime environment which can be used further referred by other CLI commands.

If -push tag is used in command then docker image is pushed to mentioned repository

*Sample config.ini file*

```
[main]
tag = $TAG
registry = $NAME_OF_REGISTRY
dockerfile = $PATH_TO_DOCKERFILE
source_folder = $MOUNT_FOLDER_PATH
; mount oci keys for local testing
oci_key_mnt = ~/.oci:/home/oci_dist_training/.oci
```

### 11.2.3.2 Publish Docker Image

**Args**

- -image: Name of the Docker image (default value is picked from config.ini file)

**Command**

```
ads opctl distributed-training publish-image
```

**Note**

This command can be used to push images to the OCI repository. In case the name of the image is not mentioned it refers to the image name from the config.ini file.

### 11.2.3.3 Run the container Image on the OCI Data Science or local

---

#### Tip

Use `-h` option to see options and usage help

```
ads opctl run -h
```

---

#### Args

- `-f`: Path to train.yaml file (required argument)
- `-b` :
  - `local` → Run DT workflow on the local environment
  - `job` → Run DT workflow on the OCI ML Jobs
  - **Note** : default value is set to jobs
- `-i`: Auto increments the tag of the image
- `-nopush`: Doesn't Push the latest image to OCIR
- `-nobuild`: Doesn't build the image
- `-t`: Tag of the docker image
- `-reg`: Docker Repository
- `-df`: Dockerfile using which docker will be build
- `-s`: source code dir

**Note** : The value “@image” for image attribute in train.yaml is replaced at runtime using combination of `-t` and `-r` params.

#### Command

##### *Local Command*

```
ads opctl run
 -f train.yaml
 -b local
 -i
```

##### *Jobs Command*

```
ads opctl run
 -f train.yaml
```

#### Note

The command `ads opctl run -f train.yaml` is used to run distributed training jobs on OCI Data Science. By default, it builds the new image and pushes it to the OCIR.

If required OCI API keys can be mounted by specifying the location in the config.ini file

### 11.2.3.4 Development Flow

#### Step 1:

Build the Docker and run it locally.

If required mount the code folder using the `-s` tag

#### Step 2:

If the user has changed files only in the mounted folder and needs to run it locally. *{Build is not required}*

```
ads opctl run
 -f train.yaml
 -b local
 -nobuild
```

In case there are some changes apart from the mounted folder and needs to run it locally. *{Build is required}*

`-i` tag is required only if the user needs to increment the tag of the image

```
ads opctl run
 -f train.yaml
 -b local
 -i
```

#### Step 3:

Finally, to run on a jobs platform

```
ads opctl run
 -f train.yaml
```

### 11.2.3.5 Diagnosing Infrastructure Setup

Before submitting your code to Data Science Jobs, check if the infra setup meets the framework requirement. Each framework has a specific set of requirements.

`ads opctl check` runs diagnosis by starting a single node `jobrun` using the container image specified in the `train.yaml` file.

```
ads opctl check -f train.yaml --output infra_report.html
```

The `train.yaml` is the same yaml file that is defined for running distributed training code. The diagnostic report is saved in the file provided in `--output` option.

Here is a sample report generated for Horovod cluster -

# Diagnostic Report for horovod

#	Requirement	Status
1	Check if job run has access to Job definition information	✓
2	Check if job run has access to VCN information	✓
3	Check if job run has access to the WORK DIR (object storage) configured in the cluster definition file	✓
4	Check Ports for Ingress	✓
5	Check if internet access is available	✓

🌟 You have satisfied all the requirements for cluster: horovod 🌟.

## 11.2.4 Dask

Dask is a flexible library for parallel computing in Python. The documentation will split between the two areas of writing distributed training using the Dask framework and creating both the container and yaml spec to run the distributed workload.

### Dask

This is a good choice when you want to use Scikit-Learn, XGBoost, LightGBM or have data parallel tasks for very large datasets where the data can be partitioned.

#### 11.2.4.1 Creating Workloads

##### Prerequisites

1. Internet Connection
2. ADS cli is [installed](#)
3. Install docker: <https://docs.docker.com/get-docker>

##### Write your training code:

While running distributed workload, the IP address of the scheduler is known only during the runtime. The IP address is exported as environment variable - SCHEDULER\_IP in all the nodes when the Job Run is in *IN\_PROGRESS* state. Create `dask.distributed.Client` object using environment variable to specify the IP address. Eg. -

```
client = Client(f"{os.environ['SCHEDULER_IP']}:{os.environ.get('SCHEDULER_PORT', '8786')}")
```

see [Writing Dask Code](#) for more examples.

For this example, the code to run on the cluster will be:

Listing 1: gridsearch.py

```

from dask.distributed import Client
from sklearn.datasets import make_classification
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

import pandas as pd
import joblib
import os
import argparse

default_n_samples = int(os.getenv("DEFAULT_N_SAMPLES", "1000"))

parser = argparse.ArgumentParser()
parser.add_argument("--n_samples", default=default_n_samples, type=int, help="size of
↳ dataset")
parser.add_argument("--cv", default=3, type=int, help="number of cross validations")
args, unknownargs = parser.parse_known_args()

Using environment variable to fetch the SCHEDULER_IP is important.
client = Client(f"{os.environ['SCHEDULER_IP']}:{os.environ.get('SCHEDULER_PORT', '8786')}
↳ ")

X, y = make_classification(n_samples=args.n_samples, random_state=42)

with joblib.parallel_backend("dask"):
 GridSearchCV(
 SVC(gamma="auto", random_state=0, probability=True),
 param_grid={
 "C": [0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0],
 "kernel": ["rbf", "poly", "sigmoid"],
 "shrinking": [True, False],
 },
 return_train_score=False,
 cv=args.cv,
 n_jobs=-1,
).fit(X, y)

```

**Initialize a distributed-training folder:**

At this point you have created a training file (or files) - `gridsearch.py` in the above example. Now running the command below

**Note:** This step requires an internet connection. The `init` command initializes your code directory with dask related artifacts to build

```
ads opctl distributed-training init --framework dask
```

**Containerize your code and build container:**

Before you can build the image, you must set the following environment variables:

Specify image name and tag

```
export IMAGE_NAME=<region.ocir.io/my-tenancy/image-name>
export TAG=latest
```

Build the container image.

```
ads opctl distributed-training build-image \
 -t $TAG \
 -reg $IMAGE_NAME \
 -df oci_dist_training_artifacts/dask/v1/Dockerfile
```

The code is assumed to be in the current working directory. To override the source code directory, use the `-s` flag and specify the code dir. This folder should be within the current working directory.

```
ads opctl distributed-training build-image \
 -t $TAG \
 -reg $IMAGE_NAME \
 -df oci_dist_training_artifacts/dask/v1/Dockerfile
 -s <code_dir>
```

If you are behind proxy, ads opctl will automatically use your proxy settings (defined via `no_proxy`, `http_proxy` and `https_proxy`).

### Define your workload yaml:

The yaml file is a declarative way to express the workload. Refer [YAML schema](#) for more details.

Listing 2: train.yaml

```
kind: distributed
apiVersion: v1.0
spec:
 infrastructure:
 kind: infrastructure
 type: dataScienceJob
 apiVersion: v1.0
 spec:
 projectId: oci.xxxx.<project_ocid>
 compartmentId: oci.xxxx.<compartment_ocid>
 displayName: my_distributed_training
 logGroupId: oci.xxxx.<log_group_ocid>
 logId: oci.xxx.<log_ocid>
 subnetId: oci.xxxx.<subnet-ocid>
 shapeName: VM.Standard2.4
 blockStorageSize: 50
 cluster:
 kind: dask
 apiVersion: v1.0
 spec:
 image: my-region.ocir.io/my-tenancy/dask-cluster-examples:dev
 workDir: "oci://my-bucket@my-namespace/daskexample/001"
 name: GridSearch Dask
 main:
 config:
 worker:
```

(continues on next page)

(continued from previous page)

```

 config:
 replicas: 2
runtime:
 kind: python
 apiVersion: v1.0
 spec:
 entryPoint: "gridsearch.py"
 kwargs: "--cv 5"
 env:
 - name: DEFAULT_N_SAMPLES
 value: 5000

```

**Use ads opctl to create the cluster infrastructure and run the workload:**

Do a dry run to inspect how the yaml translates to Job and Job Runs. This does not create actual Job or Job Run.

```
ads opctl run -f train.yaml --dry-run
```

This will give an option similar to this -

```

-----Entering dryrun mode-----
Creating Job with payload:
kind: job
spec:
 infrastructure:
 kind: infrastructure
 spec:
 blockSize: 50
 compartmentId: oci.xxxx.<compartment_ocid>
 displayName: GridSearch Dask
 jobInfrastructureType: ME_STANDALONE
 jobType: DEFAULT
 logGroupId: oci.xxxx.<log_group_ocid>
 logId: oci.xxxx.<log_ocid>
 projectId: oci.xxxx.<project_ocid>
 shapeName: VM.Standard2.4
 subnetId: oci.xxxx.<subnet-ocid>
 type: dataScienceJob
 name: GridSearch Dask
 runtime:
 kind: runtime
 spec:
 entrypoint: null
 env:
 - name: OCI__WORK_DIR
 value: oci://my-bucket@my-namespace/daskexample/001
 - name: OCI__EPHEMERAL
 value: None
 - name: OCI__CLUSTER_TYPE
 value: DASK
 - name: OCI__WORKER_COUNT
 value: '2'
 - name: OCI__START_ARGS

```

(continues on next page)

(continued from previous page)

```

 value: ''
 - name: OCI__ENTRY_SCRIPT
 value: gridsearch.py
 - name: OCI__ENTRY_SCRIPT_KWARGS
 value: --cv 5
 - name: DEFAULT_N_SAMPLES
 value: '5000'
 image: my-region.ocir.io/my-tenancy/dask-cluster-examples:dev
 type: container

+++++
Creating Main Job with following details:
Name: main
Environment Variables:
 OCI__MODE:MAIN
~~~~~
Creating 2 worker jobs with following details:
Name: worker
Environment Variables:
  OCI__MODE:WORKER
-----Ending dryrun mode-----

```

**Test Locally:**

Before submitting the workload to jobs, you can run it locally to test your code, dependencies, configurations etc. With `-b local` flag, it uses a local backend. Further when you need to run this workload on odsc jobs, simply use `-b job` flag instead.

```
ads opctl run -f train.yaml -b local
```

If your code requires to use any oci services (like object bucket), you need to mount oci keys from your local host machine onto the container. This is already done for you assuming the typical location of oci keys `~/.oci`. You can modify it though, in-case you have keys at a different location. You need to do this in the `config.ini` file.

```
oci_key_mnt = ~/.oci:/home/oci_dist_training/.oci
```

**Submit the workload:**

```
ads opctl run -f train.yaml -b job
```

**Note::** This will automatically push the docker image to the OCI [container registry repo](#) .

Once running, you will see on the terminal an output similar to the below. Note that this yaml can be used as input to `ads opctl distributed-training show-config -f <info.yaml>` - to both save and see the run info use `tee` - for example:

```
ads opctl run -f train.yaml | tee info.yaml
```

Listing 3: info.yaml

```

jobId: oci.xxxx.<job_oci>
mainJobRunId:
  mainJobRunIdName: oci.xxxx.<job_run_oci>
workDir: oci://my-bucket@my-namespace/daskcluster-testing/005

```

(continues on next page)



(continued from previous page)

**otherJobRunIds:**

- `workerJobRunIdName_1`: `oci.xxxx.<job_run_ocid>`
- `workerJobRunIdName_2`: `oci.xxxx.<job_run_ocid>`
- `workerJobRunIdName_3`: `oci.xxxx.<job_run_ocid>`

**Monitoring the workload logs**

To view the logs from a job run, you could run -

```
ads opctl watch oci.xxxx.<job_run_ocid>
```

You could stream the logs from any of the job run ocid using `ads opctl watch` command. You could run this command from multiple terminal to watch all of the job runs. Typically, watching `mainJobRunId` should yield most informative log.

To find the IP address of the scheduler dashboard, you could check the configuration file generated by the Main job by running -

```
ads opctl distributed-training show-config -f info.yaml
```

This will generate an output such as follows -

**Main Info:**

**OCI\_\_MAIN\_IP:** <ip address>

**SCHEDULER\_IP:** <ip address>

**tmpdir:** `oci://my-bucket@my-namesapce/daskcluster-testing/005/oci.xxxx.<job_ocid>`

Dask dashboard is host at : `http://{SCHEDULER_IP}:8787` If the IP address is reachable from your workstation network, you can access the dashboard directly from your workstation. The alternate approach is to use either a Bastion host on the same subnet as the Job Runs and create an ssh tunnel from your workstation.

For more information about the dashboard, checkout <https://docs.dask.org/en/stable/diagnostics-distributed.html>

**Saving Artifacts to Object Storage Buckets**

In case you want to save the artifacts generated by the training process (model checkpoints, TensorBoard logs, etc.) to an object bucket you can use the 'sync' feature. The environment variable `OCI__SYNC_DIR` exposes the directory location that will be automatically synchronized to the configured object storage bucket location. Use this directory in your training script to save the artifacts.

To configure the destination object storage bucket location, use the following settings in the workload yaml file(`train.yaml`).

- `name`: `SYNC_ARTIFACTS`  
  `value`: `1`
- `name`: `WORKSPACE`  
  `value`: `"<bucket_name>"`
- `name`: `WORKSPACE_PREFIX`  
  `value`: `"<bucket_prefix>"`

**Note:** Change `SYNC_ARTIFACTS` to `0` to disable this feature. Use `OCI__SYNC_DIR` env variable in your code to save the artifacts. For Example :

```
with open(os.path.join(os.environ.get("OCI__SYNC_DIR"), "results.txt"), "w") as rf:
    rf.write(f"Best Params are: {grid.best_params_}, Score is {grid.best_score_}")
```

### Terminating In-Progress Cluster

To terminate a running cluster, you could run -

```
ads opctl distributed-training cancel -f info.yaml
```

#### 11.2.4.2 Writing Dask Code

Dask Integrates at many levels into the Python ecosystem.

#### Run parallel computation using dask.distributed and Joblib

Joblib can use Dask as the backend. In the following example the long running function is distributed across the Dask cluster.

```
import time
import joblib

def long_running_function(i):
    time.sleep(.1)
    return i
```

This function can be called under Dask as a dask task which will be scheduled automatically by Dask across the cluster. Watching the cluster utilization will show the tasks run on the workers.

```
with joblib.parallel_backend('dask'):
    joblib.Parallel(verbose=100)(
        joblib.delayed(long_running_function)(i)
        for i in range(10))
```

#### Run parallel computation using Scikit-Learn & Joblib

To use the Dask backend to Joblib you have to create a Client, and wrap your code with the `joblib.parallel_backend('dask')` context manager.

```
import os
from dask.distributed import Client
import joblib

# the cluster once created will make available the IP address of the Dask scheduler
# through the SCHEDULER_IP environment variable
client = Client(f"{os.environ['SCHEDULER_IP']}:8786")

with joblib.parallel_backend('dask'):
    # Your scikit-learn code
```

A full example showing scaling out CPU-bound workloads; workloads with datasets that fit in RAM, but have many individual operations that can be done in parallel. To scale out to RAM-bound workloads (larger-than-memory datasets) use one of the `dask-ml` provided parallel estimators, or the `dask-ml` wrapped `XGBoost` & `LightGBM` estimators.

```
import numpy as np
from dask.distributed import Client

import joblib
from sklearn.datasets import load_digits
```

(continues on next page)

(continued from previous page)

```

from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC

client = Client(f"{os.environ['SCHEDULER_IP']}:{8786}")

digits = load_digits()

param_space = {
    'C': np.logspace(-6, 6, 13),
    'gamma': np.logspace(-8, 8, 17),
    'tol': np.logspace(-4, -1, 4),
    'class_weight': [None, 'balanced'],
}

model = SVC(kernel='rbf')
search = RandomizedSearchCV(model, param_space, cv=3, n_iter=50, verbose=10)

with joblib.parallel_backend('dask'):
    search.fit(digits.data, digits.target)

```

### 11.2.4.3 Distributed XGBoost & LightGBM

#### 11.2.4.3.1 LightGBM

For further examples and comprehensive documentation see [LightGBM](#) and [Github Examples](#)

```

import os
import joblib
import dask.array as da
from dask.distributed import Client
from sklearn.datasets import make_blobs

import lightgbm as lgb

if __name__ == "__main__":
    print("loading data")
    size = int(os.environ.get("SIZE", 1000))
    X, y = make_blobs(n_samples=size, n_features=50, centers=2)
    client = Client(
        f"{os.environ['SCHEDULER_IP']}:{os.environ.get('SCHEDULER_PORT', '8786')}"
    )

    print("distributing training data on the Dask cluster")
    dX = da.from_array(X, chunks=(100, 50))
    dy = da.from_array(y, chunks=(100,))

    print("beginning training")
    dask_model = lgb.DaskLGBMClassifier(n_estimators=10)
    dask_model.fit(dX, dy)
    assert dask_model.fitted_

```

(continues on next page)

(continued from previous page)

```
print("done training")

# Convert Dask model to sklearn model
sklearn_model = dask_model.to_local()
print(type(sklearn_model)) #<class 'lightgbm.sklearn.LGBMClassifier'>
joblib.dump(sklearn_model, "sklearn-model.joblib")
```

### 11.2.4.3.2 XGBoost

For further examples and comprehensive documentation see [XGBoost](#)

XGBoost has a Scikit-Learn interface, this provides a familiar programming interface that mimics the scikit-learn estimators with higher level of abstraction. The interface is easier to use compared to the functional interface but with more constraints. It's worth mentioning that, although the interface mimics scikit-learn estimators, it doesn't work with normal scikit-learn utilities like GridSearchCV as scikit-learn doesn't understand distributed dask data collection.

```
import os
from distributed import LocalCluster, Client
import xgboost as xgb

def main(client: Client) -> None:
    X, y = load_data()
    clf = xgb.dask.DaskXGBClassifier(n_estimators=100, tree_method="hist")
    clf.client = client # assign the client
    clf.fit(X, y, eval_set=[(X, y)])
    proba = clf.predict_proba(X)

if __name__ == "__main__":
    with Client(f"{os.environ['SCHEDULER_IP']}:8786") as client:
        main(client)
```

### 11.2.4.4 Securing with TLS

You can setup Dask cluster to run using TLS. To do so, you need three things -

1. CA Certificate
2. A Certificate signed by CA
3. Private key of the certificate

For more details refer [Dask documentation](#)

#### Self signed Certificate using openssl

openssl lets you create test CA and certificates required to setup TLS connectivity for Dask cluster. Use the commands below to create certificate in your code folder. When the container image is built, all the artifacts in the code folder is copied to /code directory inside container image.

1. Generate CA Certificate

```
openssl req -x509 -nodes -newkey rsa:4096 -days 10 -keyout dask-tls-ca-key.pem -out dask-
↪tls-ca-cert.pem -subj "/C=US/ST=CA/CN=ODSC CLUSTER PROVISIONER"
```

## 2. Generate CSR

```
openssl req -nodes -newkey rsa:4096 -keyout dask-tls-key.pem -out dask-tls-req.pem -subj
↳ "/C=US/ST=CA/CN=DASK CLUSTER"
```

## 3. Sign CSR

```
openssl x509 -req -in dask-tls-req.pem -CA dask-tls-ca-cert.pem -CAkey dask-tls-ca-key.
↳ pem -CAcreateserial -out dask-tls-cert.pem
```

4. Follow the container build instructions [here](#) to build, tag and push the image to ocir.

5. Create a cluster definition YAML and configure the certificate information under `cluster/config/startOptions`. Here is an example -

```
kind: distributed
apiVersion: v1.0
spec:
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    apiVersion: v1.0
    spec:
      projectId: oci.xxxx.<project_ocid>
      compartmentId: oci.xxxx.<compartment_ocid>
      displayName: my_distributed_training
      logGroupId: oci.xxxx.<log_group_ocid>
      logId: oci.xxx.<log_ocid>
      subnetId: oci.xxxx.<subnet-ocid>
      shapeName: VM.Standard2.4
      blockStorageSize: 50
  cluster:
    kind: dask
    apiVersion: v1.0
    spec:
      image: iad.ocir.io/mytenancy/dask-cluster-examples:dev
      workDir: oci://mybucket@mytenancy/daskexample/001
      name: LGBM Dask
      main:
        config:
          startOptions:
            - --tls-ca-file /code/dask-tls-ca-cert.pem
            - --tls-cert /code/dask-tls-cert.pem
            - --tls-key /code/dask-tls-key.pem
      worker:
        config:
          startOptions:
            - --tls-ca-file /code/dask-tls-ca-cert.pem
            - --tls-cert /code/dask-tls-cert.pem
            - --tls-key /code/dask-tls-key.pem
        replicas: 2
  runtime:
    kind: python
    apiVersion: v1.0
```

(continues on next page)

(continued from previous page)

```
spec:
  entryPoint: lgbm_dask.py
  env:
    - name: SIZE
      value: 1000000
```

### Using OCI Certificate manager

See [OCI Certificates](#) for reference. In this approach, the Admin of the tenancy or the person with the requisite permission can create and manage certificate on OCI console. Specify the OCID of the CA Certificate, TLS Certificate and Private Key of the Certificate in *cluster/certificates* option.

### Policies Required:

```
# Create DG with resource.type='certificateauthority'
```

```
Allow dynamic-group certauthority-resource to use keys in compartment <my-compartment-
↪name>
```

```
Allow dynamic-group certauthority-resource to manage objects in compartment <my-
↪compartment-name>
```

1. Create certificate authority, certificate and private key inside OCI Certificates console.
2. Create a cluster definition YAML and configure the certificate information under *cluster/config/startOptions*. Here is an example -

```
kind: distributed
apiVersion: v1.0
spec:
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    apiVersion: v1.0
    spec:
      projectId: oci.xxxx.<project_ocid>
      compartmentId: oci.xxxx.<compartment_ocid>
      displayName: my_distributed_training
      logGroupId: oci.xxxx.<log_group_ocid>
      logId: oci.xxx.<log_ocid>
      subnetId: oci.xxxx.<subnet-ocid>
      shapeName: VM.Standard2.4
      blockSizeSize: 50
  cluster:
    kind: dask
    apiVersion: v1.0
    spec:
      image: iad.ocir.io/mytenancy/dask-cluster-examples:dev
      workDir: oci://mybucket@mytenancy/daskexample/001
      name: LGBM Dask
      certificate:
        caCert:
          id: ocid1.certificateauthority.oc1.xxx.xxxxxxxx
          downloadLocation: /code/dask-tls-ca-cert.pem
        cert:
```

(continues on next page)

(continued from previous page)

```

    id: ocid1.certificate.oc1.xxx.xxxxxxx
    certDownloadLocation: /code/dask-tls-cert.pem
    keyDownloadLocation: /code/dask-tls-key.pem
  main:
    config:
      startOptions:
        - --tls-ca-file /code/dask-tls-ca-cert.pem
        - --tls-cert /code/dask-tls-cert.pem
        - --tls-key /code/dask-tls-key.pem
  worker:
    config:
      startOptions:
        - --tls-ca-file /code/dask-tls-ca-cert.pem
        - --tls-cert /code/dask-tls-cert.pem
        - --tls-key /code/dask-tls-key.pem
    replicas: 2
runtime:
  kind: python
  apiVersion: v1.0
  spec:
    entryPoint: lgbm_dask.py
    env:
      - name: SIZE
        value: 10000000

```

### 11.2.4.5 Dask Cluster Tuning

#### 11.2.4.5.1 Configuring dask startup options

##### Dask scheduler

Dask scheduler is launched with `dask-scheduler` command. By default no arguments are supplied to `dask-scheduler`. You could influence the startup option by adding them to `startOptions` under `cluster/spec/main/config` section of the cluster YAML definition

Eg. Here is how you could change the scheduler port number:

```

# Note only portion of the yaml file is shown here for brevity.
cluster:
  kind: dask
  apiVersion: v1.0
  spec:
    image: region.ocir.io/my-tenancy/image:tag
    workDir: "oci://my-bucket@my-namespaces/daskcluster-testing/005"
    ephemeral: True
    name: My Precious
    main:
      config:
        startOptions:
          - --port 8788

```

## Dask worker

Dask worker is launched with `dask-worker` command. By default no arguments are supplied to `dask-worker`. You could influence the startup option by adding them to `startOptions` under `cluster/spec/worker/config` section of the cluster YAML definition

Eg. Here is how you could change the worker port, nanny port, number of workers per host and number of threads per process:

```
# Note only portion of the yaml file is shown here for brevity.
cluster:
  kind: dask
  apiVersion: v1.0
  spec:
    image: region.ocir.io/my-tenancy/image:tag
    workDir: "oci://my-bucket@my-namespace/daskcluster-testing/005"
    ephemeral: True
    name: My Precious
    main:
      config:
    worker:
      config:
        startOptions:
          - --worker-port 8700:8800
          - --nanny-port 3000:3100
          - --nworkers 8
          - --nthreads 2
```

Refer to the complete list

### 11.2.4.5.2 Configuration through Environment Variables

You could set configuration parameters that Dask recognizes by add it to `cluster/spec/config/env` or `cluster/spec/main/config/env` or `cluster/spec/worker/config/env` If a configuration value is some for both scheduler and worker section, then set it at `cluster/spec/config/env` section.

```
# Note only portion of the yaml file is shown here for brevity.
cluster:
  kind: dask
  apiVersion: v1.0
  spec:
    image: region.ocir.io/my-tenancy/image:tag
    workDir: "oci://my-bucket@my-tenancy/daskcluster-testing/005"
    ephemeral: True
    name: My Precious
    config:
      env:
        - name: DASK_ARRAY__CHUNK_SIZE
          value: 128 MiB
        - name: DASK_DISTRIBUTED__WORKERS__MEMORY__SPILL
          value: 0.85
        - name: DASK_DISTRIBUTED__WORKERS__MEMORY__TARGET
          value: 0.75
```

(continues on next page)



(continued from previous page)

```
- name: DASK_DISTRIBUTED__WORKERS__MEMORY__TERMINATE
  value: 0.98
```

Refer [here](#) for more information

#### 11.2.4.6 Dask dashboard

Dask dashboard allows you to monitor the progress of the tasks. It gives you a real time view of the resource usage, task status, number of workers, task distribution, etc. To learn more about Dask dashboard refer [this link](#).

##### Prerequisite

1. IP address of the Main/Scheduler Node. Use `ads opctl distributed-training show-config` or find the IP address from the logs of the main job run.
2. The default port is 8787. You can override this port in `cluster/main/config/startOptions` in the cluster definition file.
3. Allow ingress to the port 8787 in the security list associated with the Subnet of Main/Scheduler node.

The dashboard is accessible over `<SCHEDULER_IP>:8787`. The IP address may not always be accessible from your workstation especially if you are using a subnet which is not connected to your corporate network. To overcome this, you could setup a bastion host on the private regional subnet that was added to the jobrun and create an ssh tunnel from your workstation to bastion host to the Job Run instance with `<SCHEDULER_IP>`

##### 11.2.4.6.1 Bastion Host

Here are the steps to setup a Bastion host to allow you to connect to the scheduler dashboard -

1. Launch a compute instance (Linux or Windows) with primary vnic with a public subnet or the subnet that is connected to your corporate network.
2. Attach a secondary VNIC on the subnet used for starting the cluster. Follow the steps detailed [here](#) on how to setup and configure the host to setup the secondary VNIC.
3. Create a public IP if you need access to the dashboard over the internet.

##### Linux instance

If you setup a Linux instance, you can create ssh tunnel from your workstation and access the scheduler dashboard from your workstation at `localhost:8787`. To setup ssh tunnel -

```
ssh -i <oci-instance-key>.key <ubuntu or opc>@<instance-ip> L 8787:<scheduler jobrun-ip>:
↪8787
```

If you are using proxy, use this command -

```
ssh -i <oci-instance-key>.key <ubuntu or opc>@<instance-ip> -o "ProxyCommand=nc -X_
↪connect -x $http_proxy:$http_port %h %p" -L 8787:<scheduler jobrun-ip>:8787
```

## Windows instance

RDP to the Windows instance and access the dashboard using <SCHEDULER\_IP>:8787 from a browser running within the Windows instance.

## 11.2.5 Horovod

### Distributed training framework for TensorFlow, Keras, PyTorch

Horovod is an open-source software framework for distributed deep learning training using TensorFlow, Keras, PyTorch. Horovod has the goal of improving the speed, scale, and resource allocation when training a machine learning model.

OCI Data Science currently support [Elastic Horovod](#) workloads with [gloo](#) backend.

#### 11.2.5.1 Creating Horovod Workloads

##### Prerequisites

1. Internet Connection
2. ADS cli is installed
3. Install docker: <https://docs.docker.com/get-docker>

##### Write your training code:

Your model training script (TensorFlow or PyTorch) needs to be adapted to use (Elastic) Horovod APIs for distributed training. Refer *[Writing distributed code with horovod framework](#)*

Also see : [Horovod Examples](#)

For this example, the code to run was inspired from an example [found here](#) . There are minimal changes to this script to save the training artifacts and TensorBoard logs to a folder referenced by OCI\_\_SYNC\_DIR environment variable. OCI\_\_SYNC\_DIR is a pre-provisioned folder which can be synchronized with an object bucket during the training process.

Listing 4: train.py

```
# Script adapted from https://github.com/horovod/horovod/blob/master/examples/elastic/
↪ tensorflow2/tensorflow2_keras_mnist_elastic.py

# =====

import argparse
import tensorflow as tf
import horovod.tensorflow.keras as hvd
from distutils.version import LooseVersion

import os

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

parser = argparse.ArgumentParser(description="Tensorflow 2.0 Keras MNIST Example")
```

(continues on next page)

(continued from previous page)

```

parser.add_argument(
    "--use-mixed-precision",
    action="store_true",
    default=False,
    help="use mixed precision for training",
)

parser.add_argument(
    "--data-dir",
    help="location of the training dataset in the local filesystem (will be downloaded,
↪if needed)",
    default='/code/data/mnist.npz'
)

args = parser.parse_args()

if args.use_mixed_precision:
    print(f"using mixed precision {args.use_mixed_precision}")
    if LooseVersion(tf.__version__) >= LooseVersion("2.4.0"):
        from tensorflow.keras import mixed_precision

        mixed_precision.set_global_policy("mixed_float16")
    else:
        policy = tf.keras.mixed_precision.experimental.Policy("mixed_float16")
        tf.keras.mixed_precision.experimental.set_policy(policy)

# Horovod: initialize Horovod.
hvd.init()

# Horovod: pin GPU to be used to process local rank (one GPU per process)
gpus = tf.config.experimental.list_physical_devices("GPU")
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], "GPU")

import numpy as np

minist_local = args.data_dir

def load_data():
    print("using pre-fetched dataset")
    with np.load(minist_local, allow_pickle=True) as f:
        x_train, y_train = f["x_train"], f["y_train"]
        x_test, y_test = f["x_test"], f["y_test"]
        return (x_train, y_train), (x_test, y_test)

(mnist_images, mnist_labels), _ = (
    load_data()
    if os.path.exists(minist_local)

```

(continues on next page)

(continued from previous page)

```

    else tf.keras.datasets.mnist.load_data(path="mnist-%d.npz" % hvd.rank())
)

dataset = tf.data.Dataset.from_tensor_slices(
    (
        tf.cast(mnist_images[...], tf.newaxis) / 255.0, tf.float32),
        tf.cast(mnist_labels, tf.int64),
    )
)
dataset = dataset.repeat().shuffle(10000).batch(128)

model = tf.keras.Sequential(
    [
        tf.keras.layers.Conv2D(32, [3, 3], activation="relu"),
        tf.keras.layers.Conv2D(64, [3, 3], activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.25),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(10, activation="softmax"),
    ]
)

# Horovod: adjust learning rate based on number of GPUs.
scaled_lr = 0.001 * hvd.size()
opt = tf.optimizers.Adam(scaled_lr)

# Horovod: add Horovod DistributedOptimizer.
opt = hvd.DistributedOptimizer(
    opt, backward_passes_per_step=1, average_aggregated_gradients=True
)

# Horovod: Specify `experimental_run_tf_function=False` to ensure TensorFlow
# uses hvd.DistributedOptimizer() to compute gradients.
model.compile(
    loss=tf.losses.SparseCategoricalCrossentropy(),
    optimizer=opt,
    metrics=["accuracy"],
    experimental_run_tf_function=False,
)

# Horovod: initialize optimizer state so we can synchronize across workers
# Keras has empty optimizer variables() for TF2:
# https://sourcegraph.com/github.com/tensorflow/tensorflow@v2.4.1/-/blob/tensorflow/
# ↪python/keras/optimizer_v2/optimizer_v2.py#L351:10
model.fit(dataset, steps_per_epoch=1, epochs=1, callbacks=None)

state = hvd.elastic.KerasState(model, batch=0, epoch=0)

```

(continues on next page)

(continued from previous page)

```

def on_state_reset():
    tf.keras.backend.set_value(state.model.optimizer.lr, 0.001 * hvd.size())
    # Re-initialize, to join with possible new ranks
    state.model.fit(dataset, steps_per_epoch=1, epochs=1, callbacks=None)

state.register_reset_callbacks([on_state_reset])

callbacks = [
    hvd.callbacks.MetricAverageCallback(),
    hvd.elastic.UpdateEpochStateCallback(state),
    hvd.elastic.UpdateBatchStateCallback(state),
    hvd.elastic.CommitStateCallback(state),
]

# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting
# them.
# save the artifacts in the OCI__SYNC_DIR dir.
artifacts_dir = os.environ.get("OCI__SYNC_DIR") + "/artifacts"
tb_logs_path = os.path.join(artifacts_dir, "logs")
check_point_path = os.path.join(artifacts_dir, "ckpts", "checkpoint-{epoch}.h5")
if hvd.rank() == 0:
    callbacks.append(tf.keras.callbacks.ModelCheckpoint(check_point_path))
    callbacks.append(tf.keras.callbacks.TensorBoard(tb_logs_path))

# Train the model.
# Horovod: adjust number of steps based on number of GPUs.
@hvd.elastic.run
def train(state):
    state.model.fit(
        dataset,
        steps_per_epoch=500 // hvd.size(),
        epochs=2 - state.epoch,
        callbacks=callbacks,
        verbose=1,
    )

train(state)

```

**Initialize a distributed-training folder:**

At this point you have created a training file (or files) - train.py from the above example. Now, run the command below.

```
ads opctl distributed-training init --framework horovod-tensorflow --version v1
```

**Note:** If you choose to run a PyTorch example instead, use horovod-pytorch as the framework.

```
ads opctl distributed-training init --framework horovod-pytorch --version v1
```

This will download the horovod-tensorflow|horovod-pytorch framework and place it inside 'oci\_dist\_training\_artifacts' folder.

**Containerize your code and build container:**

To build the image:

Horovod frameworks for TensorFlow and PyTorch contains two separate docker files, for cpu and gpu. Choose the docker file based on whether you are going to use cpu or gpu based shapes.

Before you can build the image, you must set the following environment variables:

Specify image name and tag

```
export IMAGE_NAME=<region.ocir.io/my-tenancy/image-name>
export TAG=latest
```

Build the container image.

```
ads opctl distributed-training build-image \
  -t $TAG \
  -reg $IMAGE_NAME \
  -df oci_dist_training_artifacts/horovod/v1/<pytorch|tensorflow>.<cpu|gpu>.Dockerfile
```

The code is assumed to be in the current working directory. To override the source code directory, use the `-s` flag and specify the code dir. This folder should be within the current working directory.

```
ads opctl distributed-training build-image \
  -t $TAG \
  -reg $IMAGE_NAME \
  -df oci_dist_training_artifacts/horovod/v1/<pytorch|tensorflow>.<cpu|gpu>.Dockerfile
  -s <code_dir>
```

If you are behind proxy, ads opctl will automatically use your proxy settings (defined via `no_proxy`, `http_proxy` and `https_proxy`).

**SSH Setup:**

In Horovod distributed training, communication between scheduler and worker(s) uses a secure connection. For this purpose, SSH keys need to be provisioned in the scheduler and worker nodes. This is already taken care in the docker images. When the docker image is built, SSH key pair is placed inside the image with required configuration changes (adding public key to `authorized_keys` file). This enables a secure connection between scheduler and the workers.

**Define your workload yaml:**

The yaml file is a declarative way to express the workload.

Listing 5: train.yaml

```
kind: distributed
apiVersion: v1.0
spec:
  infrastructure: # This section maps to Job definition. Does not include environment_
    ↪ variables
    kind: infrastructure
    type: dataScienceJob
    apiVersion: v1.0
    spec:
      projectId: oci.xxxx.<project_ocid>
      compartmentId: oci.xxxx.<compartment_ocid>
      displayName: HVD-Distributed-TF
```

(continues on next page)

(continued from previous page)

```

logGroupId: oci.xxxx.<log_group_ocid>
subnetId: oci.xxxx.<subnet-ocid>
shapeName: VM.GPU2.1
blockStorageSize: 50
cluster:
  kind: HOROVOD
  apiVersion: v1.0
  spec:
    image: "<region>.ocir.io/<tenancy_id>/<repo_name>/<image_name>:<image_tag>"
    workDir: "oci://<bucket_name>@<bucket_namespace>/<bucket_prefix>"
    name: "horovod_tf"
    config:
      env:
        # MIN_NP, MAX_NP and SLOTS are inferred from the shape. Modify only when
        ↪needed.
        # - name: MIN_NP
        #   value: 2
        # - name: MAX_NP
        #   value: 4
        # - name: SLOTS
        #   value: 2
        - name: WORKER_PORT
          value: 12345
        - name: START_TIMEOUT #Optional: Defaults to 600.
          value: 600
        - name: ENABLE_TIMELINE # Optional: Disabled by Default. Significantly
        ↪increases training duration if switched on (1).
          value: 0
        - name: SYNC_ARTIFACTS #Mandatory: Switched on by Default.
          value: 1
        - name: WORKSPACE #Mandatory if SYNC_ARTIFACTS==1: Destination object bucket
        ↪to sync generated artifacts to.
          value: "<bucket_name>"
        - name: WORKSPACE_PREFIX #Mandatory if SYNC_ARTIFACTS==1: Destination object
        ↪bucket folder to sync generated artifacts to.
          value: "<bucket_prefix>"
        - name: HOROVOD_ARGS # Parameters for cluster tuning.
          value: "--verbose"
      main:
        name: "scheduler"
        replicas: 1 #this will be always 1
      worker:
        name: "worker"
        replicas: 2 #number of workers
    runtime:
      kind: python
      apiVersion: v1.0
      spec:
        entryPoint: "/code/train.py" #location of user's training script in docker image.
        args: #any arguments that the training script requires.
        env:

```

Use `ads opctl` to create the cluster infrastructure and run the workload:

Do a dry run to inspect how the yaml translates to Job and Job Runs

```
ads opctl run -f train.yaml --dry-run
```

This will give output similar to this.

```
-----Entering dryrun mode-----
Creating Job with payload:
kind: job
spec:
  infrastructure:
    kind: infrastructure
    spec:
      projectId: oci.xxxx.<project_ocid>
      compartmentId: oci.xxxx.<compartment_ocid>
      displayName: HVD-Distributed-TF
      logGroupId: oci.xxxx.<log_group_ocid>
      logId: oci.xxx.<log_ocid>
      subnetId: oci.xxxx.<subnet-ocid>
      shapeName: VM.GPU2.1
      blockStorageSize: 50
    type: dataScienceJob
  name: horovod_tf
  runtime:
    kind: runtime
    spec:
      entrypoint: null
      env:
        - name: WORKER_PORT
          value: 12345
        - name: START_TIMEOUT
          value: 600
        - name: ENABLE_TIMELINE
          value: 0
        - name: SYNC_ARTIFACTS
          value: 1
        - name: WORKSPACE
          value: "<bucket_name>"
        - name: WORKSPACE_PREFIX
          value: "<bucket_prefix>"
        - name: HOROVOD_ARGS
          value: --verbose
        - name: OCI__WORK_DIR
          value: oci://<bucket_name>@<bucket_namespace>/<bucket_prefix>
        - name: OCI__EPHEMERAL
          value: None
        - name: OCI__CLUSTER_TYPE
          value: HOROVOD
        - name: OCI__WORKER_COUNT
          value: '2'
        - name: OCI__START_ARGS
          value: ''
        - name: OCI__ENTRY_SCRIPT
          value: /code/train.py
```

(continues on next page)



(continued from previous page)

```

    image: "<region>.ocir.io/<tenancy_id>/<repo_name>/<image_name>:<image_tag>"
    type: container

+++++
Creating Main Job with following details:
Name: scheduler
Environment Variables:
    OCI__MODE:MAIN
~~~~~
Creating 2 worker jobs with following details:
Name: worker
Environment Variables:
 OCI__MODE:WORKER
-----Ending dryrun mode-----

```

**Test Locally:**

Before submitting the workload to jobs, you can run it locally to test your code, dependencies, configurations etc. With `-b local` flag, it uses a local backend. Further when you need to run this workload on odsc jobs, simply use `-b job` flag instead.

```
ads opctl run -f train.yaml -b local
```

If your code requires to use any oci services (like object bucket), you need to mount oci keys from your local host machine onto the container. This is already done for you assuming the typical location of oci keys `~/.oci`. You can modify it though, in-case you have keys at a different location. You need to do this in the config.ini file.

```
oci_key_mnt = ~/.oci:/home/oci_dist_training/.oci
```

**Submit the workload:**

```
ads opctl run -f train.yaml -b job
```

**Note::** This will automatically push the docker image to the OCI container registry repo .

Once running, you will see on the terminal an output similar to the below. Note that this yaml can be used as input to `ads opctl distributed-training show-config -f <info.yaml>` - to both save and see the run info use `tee` - for example:

```
ads opctl run -f train.yaml | tee info.yaml
```

Listing 6: info.yaml

```

jobId: oci.xxxx.<job_ocid>
mainJobRunId:
 mainJobRunIdName: oci.xxxx.<job_run_ocid>
workDir: oci://my-bucket@my-namespace/daskcluster-testing/005
otherJobRunIds:
 - workerJobRunIdName_1: oci.xxxx.<job_run_ocid>
 - workerJobRunIdName_2: oci.xxxx.<job_run_ocid>
 - workerJobRunIdName_3: oci.xxxx.<job_run_ocid>

```

**Saving Artifacts to Object Storage Buckets**

In case you want to save the artifacts generated by the training process (model checkpoints, TensorBoard logs, etc.)

to an object bucket you can use the 'sync' feature. The environment variable `OCI__SYNC_DIR` exposes the directory location that will be automatically synchronized to the configured object storage bucket location. Use this directory in your training script to save the artifacts.

To configure the destination object storage bucket location, use the following settings in the workload yaml file(`train.yaml`).

```
- name: SYNC_ARTIFACTS
 value: 1
- name: WORKSPACE
 value: "<bucket_name>"
- name: WORKSPACE_PREFIX
 value: "<bucket_prefix>"
```

**Note:** Change `SYNC_ARTIFACTS` to `0` to disable this feature. Use `OCI__SYNC_DIR` env variable in your code to save the artifacts. For Example :

```
tf.keras.callbacks.ModelCheckpoint(os.path.join(os.environ.get("OCI__SYNC_DIR"), "ckpts",
↪ 'checkpoint-{epoch}.h5'))
```

### Monitoring the workload logs

To view the logs from a job run, you could run -

```
ads jobs watch oci.xxxx.<job_run_ocid>
```

For more monitoring options, please refer to [Monitoring Horovod Training](#)

## 11.2.5.2 Writing Distributed code with Horovod Framework

### 11.2.5.2.1 TensorFlow

To use Horovod in TensorFlow, following modifications are required in the training script:

1. Import Horovod and initialize it.

```
import horovod.tensorflow as hvd
hvd.init()
```

2. Pin each GPU to a single process.

With **TensorFlow v1**.

```
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())
```

With **TensorFlow v2**.

```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
 tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
 tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

3. Scale the learning rate by the number of workers.

```
opt = tf.keras.optimizers.SGD(0.0005 * hvd.size())
```

4. Wrap the optimizer in `hvd.DistributedOptimizer`.

```
opt = hvd.DistributedOptimizer(opt)
```

5. Modify your code to save checkpoints(and any other artifacts) only in the rank-0 training process to prevent other workers from corrupting them.

```
if hvd.rank() == 0:
 tf.keras.callbacks.ModelCheckpoint(ckpts_path)
 tf.keras.callbacks.TensorBoard(tblogs_path)
```

6. OCI Data Science Horovod workloads are based on Elastic Horovod. In addition to above changes, the training script also needs to use `state synchronization`. In summary, this means:

- a. Use the decorator `hvd.elastic.run` to wrap the main training process.
- b. Use `hvd.elastic.State` to add all variables that needs to be sync across workers.
- c. Save state periodically, using `hvd.elastic.State`

A complete example can be found in the [Write your training code](#) section. More examples can be found [here](#). Refer [horovod with TensorFlow](#) and [horovod with Keras](#) for more details.

### 11.2.5.2.2 PyTorch

To use Horovod in PyTorch, following modifications are required in the training script:

1. Import Horovod and initialize it.

```
import horovod.torch as hvd
hvd.init()
```

2. Pin each GPU to a single process. (use `hvd.local_rank()`)

```
torch.manual_seed(args.seed)
if args.cuda:
 # Horovod: pin GPU to local rank.
 torch.cuda.set_device(hvd.local_rank())
 torch.cuda.manual_seed(args.seed)
```

3. Scale the learning rate by the number of workers. (use `hvd.size()`)

```
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
 momentum=args.momentum)
```

4. Wrap the optimizer in `hvd.DistributedOptimizer`.

```
optimizer = hvd.DistributedOptimizer(
 optimizer,
 named_parameters=model.named_parameters(),
 compression=compression,
 op=hvd.Adasum if args.use_adasum else hvd.Average
)
```

5. Modify your code to save checkpoints only in the rank-0 training process to prevent other workers from corrupting them.
6. Like TensorFlow, Horovod PyTorch scripts also need to use [state synchronization](#). Refer TensorFlow section [above](#). Here is a complete PyTorch sample which is inspired from examples found [here](#) and [here](#).

Listing 7: train.py

```
Script adapted from https://github.com/horovod/horovod/blob/master/examples/elastic/
↳ pytorch/pytorch_mnist_elastic.py

=====
import argparse
import os
from filelock import FileLock

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torch.utils.data.distributed
import horovod.torch as hvd
from torch.utils.tensorboard import SummaryWriter

Training settings
parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
parser.add_argument('--batch-size', type=int, default=64, metavar='N',
 help='input batch size for training (default: 64)')
parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
 help='input batch size for testing (default: 1000)')
parser.add_argument('--epochs', type=int, default=10, metavar='N',
 help='number of epochs to train (default: 10)')
parser.add_argument('--lr', type=float, default=0.01, metavar='LR',
 help='learning rate (default: 0.01)')
parser.add_argument('--momentum', type=float, default=0.5, metavar='M',
 help='SGD momentum (default: 0.5)')
parser.add_argument('--no-cuda', action='store_true', default=False,
 help='disables CUDA training')
parser.add_argument('--seed', type=int, default=42, metavar='S',
 help='random seed (default: 42)')
parser.add_argument('--log-interval', type=int, default=10, metavar='N',
 help='how many batches to wait before logging training status')
parser.add_argument('--fp16-allreduce', action='store_true', default=False,
 help='use fp16 compression during allreduce')
parser.add_argument('--use-adasum', action='store_true', default=False,
 help='use adasum algorithm to do reduction')
parser.add_argument('--data-dir',
 help='location of the training dataset in the local filesystem (will_
↳ be downloaded if needed)')

args = parser.parse_args()
args.cuda = not args.no_cuda and torch.cuda.is_available()
```

(continues on next page)

(continued from previous page)

```

checkpoint_format = 'checkpoint-{epoch}.pth.tar'

Horovod: initialize library.
hvd.init()
torch.manual_seed(args.seed)

if args.cuda:
 # Horovod: pin GPU to local rank.
 torch.cuda.set_device(hvd.local_rank())
 torch.cuda.manual_seed(args.seed)

Horovod: limit # of CPU threads to be used per worker.
torch.set_num_threads(1)

kwargs = {'num_workers': 1, 'pin_memory': True} if args.cuda else {}
data_dir = args.data_dir or './data'
with FileLock(os.path.expanduser("~/horovod_lock")):
 train_dataset = \
 datasets.MNIST(data_dir, train=True, download=True,
 transform=transforms.Compose([
 transforms.ToTensor(),
 transforms.Normalize((0.1307,), (0.3081,))
]))
 # Horovod: use DistributedSampler to partition the training data.
 train_sampler = torch.utils.data.distributed.DistributedSampler(
 train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
 train_loader = torch.utils.data.DataLoader(
 train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)

test_dataset = \
 datasets.MNIST(data_dir, train=False, transform=transforms.Compose([
 transforms.ToTensor(),
 transforms.Normalize((0.1307,), (0.3081,))
]))
 # Horovod: use DistributedSampler to partition the test data.
 test_sampler = torch.utils.data.distributed.DistributedSampler(
 test_dataset, num_replicas=hvd.size(), rank=hvd.rank())
 test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=args.test_batch_size,
 sampler=test_sampler, **kwargs)

class Net(nn.Module):
 def __init__(self):
 super(Net, self).__init__()
 self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
 self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
 self.conv2_drop = nn.Dropout2d()
 self.fc1 = nn.Linear(320, 50)
 self.fc2 = nn.Linear(50, 10)

 def forward(self, x):

```

(continues on next page)

(continued from previous page)

```

 x = F.relu(F.max_pool2d(self.conv1(x), 2))
 x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
 x = x.view(-1, 320)
 x = F.relu(self.fc1(x))
 x = F.dropout(x, training=self.training)
 x = self.fc2(x)
 return F.log_softmax(x)

model = Net()

By default, Adasum doesn't need scaling up learning rate.
lr_scaler = hvd.size() if not args.use_adasum else 1

if args.cuda:
 # Move model to GPU.
 model.cuda()
 # If using GPU Adasum allreduce, scale learning rate by local_size.
 if args.use_adasum and hvd.nccl_built():
 lr_scaler = hvd.local_size()

Horovod: scale learning rate by lr_scaler.
optimizer = optim.SGD(model.parameters(), lr=args.lr * lr_scaler,
 momentum=args.momentum)

Horovod: (optional) compression algorithm.
compression = hvd.Compression.fp16 if args.fp16_allreduce else hvd.Compression.none

def metric_average(val, name):
 tensor = torch.tensor(val)
 avg_tensor = hvd.allreduce(tensor, name=name)
 return avg_tensor.item()

def create_dir(dir):
 if not os.path.exists(dir):
 os.makedirs(dir)
Horovod: average metrics from distributed training.
class Metric(object):
 def __init__(self, name):
 self.name = name
 self.sum = torch.tensor(0.)
 self.n = torch.tensor(0.)

 def update(self, val):
 self.sum += hvd.allreduce(val.detach().cpu(), name=self.name)
 self.n += 1

 @property
 def avg(self):
 return self.sum / self.n

```

(continues on next page)

(continued from previous page)

```

@hvd.elastic.run
def train(state):
 # post synchronization event (worker added, worker removed) init ...

 artifacts_dir = os.environ.get("OCI__SYNC_DIR") + "/artifacts"
 chkpts_dir = os.path.join(artifacts_dir, "ckpts")
 logs_dir = os.path.join(artifacts_dir, "logs")
 if hvd.rank() == 0:
 print("creating dirs for checkpoints and logs")
 create_dir(chkpts_dir)
 create_dir(logs_dir)

 writer = SummaryWriter(logs_dir) if hvd.rank() == 0 else None

 for state.epoch in range(state.epoch, args.epochs + 1):
 train_loss = Metric('train_loss')
 state.model.train()

 train_sampler.set_epoch(state.epoch)
 steps_remaining = len(train_loader) - state.batch

 for state.batch, (data, target) in enumerate(train_loader):
 if state.batch >= steps_remaining:
 break

 if args.cuda:
 data, target = data.cuda(), target.cuda()
 state.optimizer.zero_grad()
 output = state.model(data)
 loss = F.nll_loss(output, target)
 train_loss.update(loss)
 loss.backward()
 state.optimizer.step()
 if state.batch % args.log_interval == 0:
 # Horovod: use train_sampler to determine the number of examples in
 # this worker's partition.
 print('Train Epoch: {} [{} / {} ({:.0f}%)] \t Loss: {:.6f}'.format(
 state.epoch, state.batch * len(data), len(train_sampler),
 100.0 * state.batch / len(train_loader), loss.item()))
 state.commit()
 if writer:
 writer.add_scalar("Loss", train_loss.avg, state.epoch)
 if hvd.rank() == 0:
 chkpt_path = os.path.join(chkpts_dir, checkpoint_format.format(epoch=state.
↪ epoch + 1))
 chkpt = {
 'model': state.model.state_dict(),
 'optimizer': state.optimizer.state_dict(),
 }
 torch.save(chkpt, chkpt_path)
 state.batch = 0

```

(continues on next page)

(continued from previous page)

```

def test():
 model.eval()
 test_loss = 0.
 test_accuracy = 0.
 for data, target in test_loader:
 if args.cuda:
 data, target = data.cuda(), target.cuda()
 output = model(data)
 # sum up batch loss
 test_loss += F.nll_loss(output, target, size_average=False).item()
 # get the index of the max log-probability
 pred = output.data.max(1, keepdim=True)[1]
 test_accuracy += pred.eq(target.data.view_as(pred)).cpu().float().sum()

 # Horovod: use test_sampler to determine the number of examples in
 # this worker's partition.
 test_loss /= len(test_sampler)
 test_accuracy /= len(test_sampler)

 # Horovod: average metric values across workers.
 test_loss = metric_average(test_loss, 'avg_loss')
 test_accuracy = metric_average(test_accuracy, 'avg_accuracy')

 # Horovod: print output only on first rank.
 if hvd.rank() == 0:
 print('\nTest set: Average loss: {:.4f}, Accuracy: {:.2f}%\n'.format(
 test_loss, 100. * test_accuracy))

Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(optimizer,
 named_parameters=model.named_parameters(),
 compression=compression,
 op=hvd.Adasum if args.use_adasum else hvd.Average)

adjust learning rate on reset
def on_state_reset():
 for param_group in optimizer.param_groups:
 param_group['lr'] = args.lr * hvd.size()

state = hvd.elastic.TorchState(model, optimizer, epoch=1, batch=0)
state.register_reset_callbacks([on_state_reset])
train(state)
test()

```

Refer to more examples [here](#). Refer [horovod with PyTorch](#) for more details.

### Next Steps

Once you have the training code ready (either in TensorFlow or PyTorch), you can proceed to [creating Horovod workloads](#).



### 11.2.5.3 Monitoring Training

Monitoring Horovod training using TensorBoard is similar to how it is usually done for TensorFlow or PyTorch workloads. Your training script generates the TensorBoard logs and saves the logs to the directory reference by `OCI__SYNC_DIR` env variable. With `SYNC_ARTIFACTS=1`, these TensorBoard logs will be periodically synchronized with the configured object storage bucket.

Please refer *[Saving Artifacts to Object Storage Buckets](#)*.

#### Aggregating metrics:

In a distributed setup, the metrics(loss, accuracy etc.) need to be aggregated from all the workers. Horovod provides [MetricAverageCallback](#) callback(for TensorFlow) which should be added to the model training step. For PyTorch, refer this [Pytorch Example](#).

#### Using TensorBoard Logs:

TensorBoard can be setup on a local machine and pointed to object storage. This will enable a live monitoring setup of TensorBoard logs.

```
OCI__IAM_TYPE=api_key tensorboard --logdir oci://<bucket_name>/path/to/logs
```

**Note:** The logs take some initial time (few minutes) to reflect on the tensorboard dashboard.

#### Horovod Timelines:

Horovod also provides [Timelines](#), which provides a snapshot of the training activities. Timeline files can be optionally generated with the following environment variable(part of workload yaml).

```
config:
 env:
 - name: ENABLE_TIMELINE #Disabled by Default(0).
 value: 1
```

**Note:** Creating Timelines degrades the training execution time.

### 11.2.6 PyTorch Distributed

PyTorch is an open source machine learning framework used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab. ADS supports running PyTorch's native distributed training code (`torch.distributed` and `DistributedDataParallel`) with OCI Data Science Jobs. Provided you are following the [official PyTorch distributed data parallel guidelines](#), **no changes to your PyTorch code are required**.

PyTorch distributed training requires initialization using the `torch.distributed.init_process_group()` function. By default this function collects uses environment variables to initialize the communications for the training cluster. When using ADS to run PyTorch distributed training on OCI data science Jobs, the environment variables, including `MASTER_ADDR`, `MASTER_PORT`, `WORLD_SIZE` RANK, and `LOCAL_RANK` will automatically be set in the job runs. By default `MASTER_PORT` will be set to 29400.

### 11.2.6.1 Creating PyTorch Distributed Workloads

#### Prerequisites

1. Internet Connection
2. ADS cli is installed
3. Install docker: <https://docs.docker.com/get-docker>

#### Write your training code:

For this example, the code to run was inspired from an example [found here](#)

Note that MASTER\_ADDR, MASTER\_PORT, WORLD\_SIZE, RANK, and LOCAL\_RANK are environment variables that will automatically be set.

Listing 8: train.py

```
Copyright (c) 2017 Facebook, Inc. All rights reserved.
BSD 3-Clause License
#
Script adapted from:
https://github.com/Azure/azureml-examples/blob/main/python-sdk/workflows/train/
→pytorch/cifar-distributed/src/train.py
=====

import datetime
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import os, argparse

define network architecture
class Net(nn.Module):
 def __init__(self):
 super(Net, self).__init__()
 self.conv1 = nn.Conv2d(3, 32, 3)
 self.pool = nn.MaxPool2d(2, 2)
 self.conv2 = nn.Conv2d(32, 64, 3)
 self.conv3 = nn.Conv2d(64, 128, 3)
 self.fc1 = nn.Linear(128 * 6 * 6, 120)
 self.dropout = nn.Dropout(p=0.2)
 self.fc2 = nn.Linear(120, 84)
 self.fc3 = nn.Linear(84, 10)

 def forward(self, x):
 x = F.relu(self.conv1(x))
 x = self.pool(F.relu(self.conv2(x)))
 x = self.pool(F.relu(self.conv3(x)))
 x = x.view(-1, 128 * 6 * 6)
 x = self.dropout(F.relu(self.fc1(x)))
```

(continues on next page)

(continued from previous page)

```

 x = F.relu(self.fc2(x))
 x = self.fc3(x)
 return x

define functions
def train(train_loader, model, criterion, optimizer, epoch, device, print_freq, rank):
 running_loss = 0.0
 for i, data in enumerate(train_loader, 0):
 # get the inputs; data is a list of [inputs, labels]
 inputs, labels = data[0].to(device), data[1].to(device)

 # zero the parameter gradients
 optimizer.zero_grad()

 # forward + backward + optimize
 outputs = model(inputs)
 loss = criterion(outputs, labels)
 loss.backward()
 optimizer.step()

 # print statistics
 running_loss += loss.item()
 if i % print_freq == 0: # print every print_freq mini-batches
 print(
 "Rank %d: [%d, %5d] loss: %.3f"
 % (rank, epoch + 1, i + 1, running_loss / print_freq)
)
 running_loss = 0.0

def evaluate(test_loader, model, device):
 classes = (
 "plane",
 "car",
 "bird",
 "cat",
 "deer",
 "dog",
 "frog",
 "horse",
 "ship",
 "truck",
)

 model.eval()

 correct = 0
 total = 0
 class_correct = list(0.0 for i in range(10))
 class_total = list(0.0 for i in range(10))
 with torch.no_grad():

```

(continues on next page)

(continued from previous page)

```

 for data in test_loader:
 images, labels = data[0].to(device), data[1].to(device)
 outputs = model(images)
 _, predicted = torch.max(outputs.data, 1)
 total += labels.size(0)
 correct += (predicted == labels).sum().item()
 c = (predicted == labels).squeeze()
 for i in range(10):
 label = labels[i]
 class_correct[label] += c[i].item()
 class_total[label] += 1

print total test set accuracy
print(
 "Accuracy of the network on the 10000 test images: %d %"
 % (100 * correct / total)
)

print test accuracy for each of the classes
for i in range(10):
 print(
 "Accuracy of %5s : %2d %"
 % (classes[i], 100 * class_correct[i] / class_total[i])
)

def main(args):
 # get PyTorch environment variables
 world_size = int(os.environ["WORLD_SIZE"])
 rank = int(os.environ["RANK"])
 local_rank = int(os.environ["LOCAL_RANK"])

 distributed = world_size > 1

 if torch.cuda.is_available():
 print("CUDA is available.")
 else:
 print("CUDA is not available.")

 # set device
 if distributed:
 if torch.cuda.is_available():
 device = torch.device("cuda", local_rank)
 else:
 device = torch.device("cpu")
 else:
 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

 # initialize distributed process group using default env:// method
 if distributed:
 torch.distributed.init_process_group(
 backend=args.backend,

```

(continues on next page)

(continued from previous page)

```

 timeout=datetime.timedelta(minutes=args.timeout)
)

 # define train and test dataset DataLoaders
 transform = transforms.Compose(
 [transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]
)

 train_set = torchvision.datasets.CIFAR10(
 root=args.data_dir, train=True, download=True, transform=transform
)

 if distributed:
 train_sampler = torch.utils.data.distributed.DistributedSampler(train_set)
 else:
 train_sampler = None

 train_loader = torch.utils.data.DataLoader(
 train_set,
 batch_size=args.batch_size,
 shuffle=(train_sampler is None),
 num_workers=args.workers,
 sampler=train_sampler,
)

 test_set = torchvision.datasets.CIFAR10(
 root=args.data_dir, train=False, download=True, transform=transform
)
 test_loader = torch.utils.data.DataLoader(
 test_set, batch_size=args.batch_size, shuffle=False, num_workers=args.workers
)

 model = Net().to(device)

 # wrap model with DDP
 if distributed:
 if torch.cuda.is_available():
 model = nn.parallel.DistributedDataParallel(
 model, device_ids=[local_rank], output_device=local_rank
)
 else:
 model = nn.parallel.DistributedDataParallel(model)

 # define loss function and optimizer
 criterion = nn.CrossEntropyLoss()
 optimizer = optim.SGD(
 model.parameters(), lr=args.learning_rate, momentum=args.momentum
)

 # train the model
 for epoch in range(args.epochs):
 print("Rank %d: Starting epoch %d" % (rank, epoch))

```

(continues on next page)

(continued from previous page)

```

 if distributed:
 train_sampler.set_epoch(epoch)
 model.train()
 train(
 train_loader,
 model,
 criterion,
 optimizer,
 epoch,
 device,
 args.print_freq,
 rank,
)

 print("Rank %d: Finished Training" % (rank))

 if not distributed or rank == 0:
 os.makedirs(args.output_dir, exist_ok=True)
 model_path = os.path.join(args.output_dir, "cifar_net.pt")
 torch.save(model.state_dict(), model_path)

 # evaluate on full test dataset
 evaluate(test_loader, model, device)

run script
if __name__ == "__main__":
 # setup argparse
 parser = argparse.ArgumentParser()
 parser.add_argument(
 "--data-dir", type=str, help="directory containing CIFAR-10 dataset"
)
 parser.add_argument("--epochs", default=10, type=int, help="number of epochs")
 parser.add_argument(
 "--batch-size",
 default=16,
 type=int,
 help="mini batch size for each gpu/process",
)
 parser.add_argument(
 "--workers",
 default=2,
 type=int,
 help="number of data loading workers for each gpu/process",
)
 parser.add_argument(
 "--learning-rate", default=0.001, type=float, help="learning rate"
)
 parser.add_argument("--momentum", default=0.9, type=float, help="momentum")
 parser.add_argument(
 "--output-dir", default="outputs", type=str, help="directory to save model to"
)

```

(continues on next page)

(continued from previous page)

```

parser.add_argument(
 "--print-freq",
 default=200,
 type=int,
 help="frequency of printing training statistics",
)
parser.add_argument(
 "--backend", default="gloo", type=str,
 help="distributed communication backend, should be gloo, nccl or mpi"
)
parser.add_argument(
 "--timeout", default=30, type=int,
 help="timeout in minutes for waiting for the initialization of distributed_
↪process group."
)
args = parser.parse_args()

call main function
main(args)

```

**Initialize a distributed-training folder:**

At this point you have create a training file (or files) - `train.py` in the above example. Now running the command below will download the artifacts required for building the docker image. The artifacts will be saved into the `oci_dist_training_artifacts/pytorch/v1` directory under your current working directory.

```
ads opctl distributed-training init --framework pytorch --version v1
```

**Containerize your code and build container:**

Before you can build the image, you must set the following environment variables:

Specify image name and tag

```
export IMAGE_NAME=<region.ocir.io/my-tenancy/image-name>
export TAG=latest
```

Build the container image.

```
ads opctl distributed-training build-image \
-t $TAG \
-reg $IMAGE_NAME \
-df oci_dist_training_artifacts/pytorch/v1/Dockerfile
```

The code is assumed to be in the current working directory. To override the source code directory, use the `-s` flag and specify the code dir. This folder should be within the current working directory.

```
ads opctl distributed-training build-image \
-t $TAG \
-reg $IMAGE_NAME \
-df oci_dist_training_artifacts/horovod/v1/oci_dist_training_artifacts/pytorch/v1/
↪Dockerfile
-s <code_dir>
```

If you are behind proxy, `ads opctl` will automatically use your proxy settings (defined via `no_proxy`, `http_proxy` and `https_proxy`).

### Define your workload yaml:

The `yaml` file is a declarative way to express the workload. Following is the YAML for running the example code, you will need to replace the values in the *spec* sections for your project:

- `infrastructure` contains *spec* for OCI Data Science Jobs. Here you need to specify a subnet that allows communications between nodes. The `VM.GPU2.1` shape is used in this example.
- `cluster` contains *spec* for the image you built and a working directory on OCI object storage, which will be used by job runs to shared internal configurations. Environment variables specified in the `cluster.spec.config` will be available in all nodes. Here the `NCCL_ASYNC_ERROR_HANDLING` is used to enable the timeout for NCCL backend. The job runs will be terminated if the nodes failed to connect to each other in certain minutes as specified in your training code when calling `init_process_group()`.
- `runtime` contains *spec* for the name of your training script, and the command line arguments for running the script. Here the `nccl` backend is used for communications between GPUs. For CPU training, you can use the `gloo` backend. The `timeout` argument specify the maximum minutes for the nodes to wait when calling `init_process_group()`. This is useful for preventing the job runs to wait forever in case of node failure.

Listing 9: `train.yaml`

```
kind: distributed
apiVersion: v1.0
spec:
 infrastructure:
 kind: infrastructure
 type: dataScienceJob
 apiVersion: v1.0
 spec:
 projectId: oci.xxxx.<project_ocid>
 compartmentId: oci.xxxx.<compartment_ocid>
 displayName: PyTorch-Distributed
 logGroupId: oci.xxxx.<log_group_ocid>
 logId: oci.xxx.<log_ocid>
 subnetId: oci.xxxx.<subnet-ocid>
 shapeName: VM.GPU2.1
 blockSizeSize: 50
 cluster:
 kind: pytorch
 apiVersion: v1.0
 spec:
 image: <region.ocir.io/my-tenancy/image-name>
 workDir: "oci://my-bucket@my-namespace/pytorch/distributed"
 config:
 env:
 - name: NCCL_ASYNC_ERROR_HANDLING
 value: '1'
 main:
 name: PyTorch-Distributed-main
 replicas: 1
 worker:
 name: PyTorch-Distributed-worker
 replicas: 3
```

(continues on next page)



(continued from previous page)

```
runtime:
 kind: python
 apiVersion: v1.0
 spec:
 entryPoint: "train.py"
 args:
 - --data-dir
 - /home/datascience/data
 - --output-dir
 - /home/datascience/outputs
 - --backend
 - gloo
 - --timeout
 - 5
```

Use `ads opctl` to create the cluster infrastructure and dry-run the workload:

```
ads opctl run -f train.yaml --dry-run
```

the output from the dry run will show all the actions and infrastructure configuration.

Use `ads opctl` to create the cluster infrastructure and run the workload:

#### Test Locally:

Before submitting the workload to jobs, you can run it locally to test your code, dependencies, configurations etc. With `-b local` flag, it uses a local backend. Further when you need to run this workload on odsc jobs, simply use `-b job` flag instead.

```
ads opctl run -f train.yaml -b local
```

If your code requires to use any oci services (like object bucket), you need to mount oci keys from your local host machine onto the container. This is already done for you assuming the typical location of oci keys `~/.oci`. You can modify it though, in-case you have keys at a different location. You need to do this in the `config.ini` file.

```
oci_key_mnt = ~/.oci:/home/oci_dist_training/.oci
```

#### Submit the workload:

```
ads opctl run -f train.yaml -b job
```

**Note:** This will automatically push the docker image to the OCI [container registry repo](#).

Once running, you will see on the terminal an output similar to the below. Note that this yaml can be used as input to `ads opctl distributed-training show-config -f <info.yaml>` - to both save and see the run info use `tee` - for example:

```
ads opctl run -f train.yaml | tee info.yaml
```

Listing 10: info.yaml

```
jobId: oci.xxxx.<job_oci>
mainJobRunId:
 mainJobRunIdName: oci.xxxx.<job_run_oci>
workDir: oci://my-bucket@my-namespace/daskcluster-testing/005
```

(continues on next page)

(continued from previous page)

**otherJobRunIds:**

- `workerJobRunIdName_1`: oci.xxxx.<job\_run\_ocid>
- `workerJobRunIdName_2`: oci.xxxx.<job\_run\_ocid>
- `workerJobRunIdName_3`: oci.xxxx.<job\_run\_ocid>

**Saving Artifacts to Object Storage Buckets**

In case you want to save the artifacts generated by the training process (model checkpoints, TensorBoard logs, etc.) to an object bucket you can use the 'sync' feature. The environment variable `OCI__SYNC_DIR` exposes the directory location that will be automatically synchronized to the configured object storage bucket location. Use this directory in your training script to save the artifacts.

To configure the destination object storage bucket location, use the following settings in the workload yaml file(train.yaml).

```
- name: SYNC_ARTIFACTS
 value: 1
- name: WORKSPACE
 value: "<bucket_name>"
- name: WORKSPACE_PREFIX
 value: "<bucket_prefix>"
```

**Note:** Change `SYNC_ARTIFACTS` to 0 to disable this feature. Use `OCI__SYNC_DIR` env variable in your code to save the artifacts. For Example :

```
model_path = os.path.join(os.environ.get("OCI__SYNC_DIR"), "model.pt")
torch.save(model, model_path)
```

**Profiling**

You may want to profile your training setup for optimization/performance tuning. Profiling typically provides a detailed analysis of cpu utilization, gpu utilization, top cuda kernels, top operators etc. You can choose to profile your training setup using the native Pytorch profiler or using a third party profiler such as [Nvidia Nsights](#).

**Profiling using Pytorch Profiler**

Pytorch Profiler is a native offering from Pytorch for Pytorch performance profiling. Profiling is invoked using code instrumentation using the api `torch.profiler.profile`.

Refer [this link](#) for changes that you need to do in your training script for instrumentation. You should choose the `OCI__SYNC_DIR` directory to save the profiling logs. For example:

```
prof = torch.profiler.profile(activities=[torch.profiler.ProfilerActivity.CPU, torch.
↳ profiler.ProfilerActivity.CUDA],
 schedule=torch.profiler.schedule(
 wait=1,
 warmup=1,
 active=3,
 repeat=1),
 on_trace_ready=torch.profiler.tensorboard_trace_handler(os.environ.get("OCI__SYNC_
↳ DIR") + "/logs"),
 with_stack=False)
prof.start()

training code
prof.end()
```

Also, the sync feature SYNC\_ARTIFACTS should be enabled '1' to sync the profiling logs to the configured object storage.

You would also need to install the Pytorch Tensorboard Plugin.

```
pip install torch-tb-profiler
```

Thereafter, use Tensorboard to view logs. Refer the [Tensorboard setup](#) for set-up on your computer.

### Profiling using Nvidia Nsights

[Nvidia Nsights](#), is a system wide profiling tool from Nvidia that can be used to profile Deep Learning workloads.

Nsights requires no change in your training code. This works on process level. You can enable this experimental feature in your training setup via the following configuration in the runtime yaml file(highlighted).

```
spec:
 image: "@image"
 workDir: "oci://@"
 name: "tf_multiworker"
 config:
 env:
 - name: WORKER_PORT
 value: 12345
 - name: SYNC_ARTIFACTS
 value: 1
 - name: WORKSPACE
 value: "<bucket_name>"
 - name: WORKSPACE_PREFIX
 value: "<bucket_prefix>"
 - name: PROFILE
 value: 1
 - name: PROFILE_CMD
 value: "nsys profile -w true -t cuda,nvtx,osrt,cudnn,cublas -s none -o /opt/
ml/nsight_report -x true"
 main:
 name: "main"
 replicas: 1
 worker:
 name: "worker"
 replicas: 1
```

Refer [this](#) for nsys profile command options. You can modify the command within the PROFILE\_CMD but remember this is all experimental. The profiling reports are generated per node. You need to download the reports to your computer manually or via the oci command.

```
oci os object bulk-download \
 -ns <namespace> \
 -bn <bucket_name> \
 --download-dir /path/on/your/computer \
 --prefix path/on/bucket/<job_id>
```

**Note:** -bn == WORKSPACE and --prefix path == WORKSPACE\_PREFIX/<job\_id> , as configured in the runtime yaml file. To view the reports, you would need to install Nsight Systems app from [here](#). Thereafter, open the downloaded reports in the Nsight Systems app.

## 11.2.7 Tensorflow

### Distributed training with Native TensorFlow

TensorFlow is an open-source software framework for distributed deep learning training. Tensorflow has multiple strategies. The following are supported:

1. MirroredStrategy
2. MultiWorkerMirroredStrategy
3. ParameterServerStrategy

#### 11.2.7.1 Creating Tensorflow Workloads

##### Prerequisites

1. Internet Connection
2. ADS cli is installed
3. Install docker: <https://docs.docker.com/get-docker>

##### Write your training code:

Your model training script needs to use one of Distributed Strategies in tensorflow.

For example, you can have the following training Tensorflow script for MultiWorkerMirroredStrategy saved as *mnist.py*:

```
Script adapted from tensorflow tutorial: https://www.tensorflow.org/tutorials/
↪distribute/multi_worker_with_keras
import tensorflow as tf
import tensorflow_datasets as tfds
import os
import sys
import time
import ads
from ocifs import OCIFileSystem
from tensorflow.data.experimental import AutoShardPolicy

BUFFER_SIZE = 10000
BATCH_SIZE_PER_REPLICA = 64

if '.' not in sys.path:
 sys.path.insert(0, '.')

def create_dir(dir):
 if not os.path.exists(dir):
 os.makedirs(dir)

def create_dirs(task_type="worker", task_id=0):
 artifacts_dir = os.environ.get("OCI__SYNC_DIR", "/opt/ml")
 model_dir = artifacts_dir + "/model"
 print("creating dirs for Model: ", model_dir)
 create_dir(model_dir)
 checkpoint_dir = write_filepath(artifacts_dir, task_type, task_id)
```

(continues on next page)

(continued from previous page)

```

 return artifacts_dir, checkpoint_dir, model_dir

def write_filepath(artifacts_dir, task_type, task_id):
 if task_type == None:
 task_type = "worker"
 checkpoint_dir = artifacts_dir + "/checkpoints/" + task_type + "/" + str(task_id)
 print("creating dirs for Checkpoints: ", checkpoint_dir)
 create_dir(checkpoint_dir)
 return checkpoint_dir

def scale(image, label):
 image = tf.cast(image, tf.float32)
 image /= 255
 return image, label

def get_data(data_bckt=None, data_dir="/code/data", num_replicas=1, num_workers=1):
 if data_bckt is not None and not os.path.exists(data_dir + '/mnist'):
 print(f"downloading data from {data_bckt}")
 ads.set_auth(os.environ.get("OCI_IAM_TYPE", "resource_principal"))
 authinfo = ads.common.auth.default_signer()
 oci_filesystem = OCIFFileSystem(**authinfo)
 lck_file = os.path.join(data_dir, '.lck')
 if not os.path.exists(lck_file):
 os.makedirs(os.path.dirname(lck_file), exist_ok=True)
 open(lck_file, 'w').close()
 oci_filesystem.download(data_bckt, data_dir, recursive=True)
 else:
 print(f"data downloaded by a different process. waiting")
 time.sleep(30)

 BATCH_SIZE = BATCH_SIZE_PER_REPLICA * num_replicas * num_workers
 print("Now printing data_dir:", data_dir)
 datasets, info = tfds.load(name='mnist', with_info=True, as_supervised=True, data_dir=data_dir)
 mnist_train, mnist_test = datasets['train'], datasets['test']
 print("num_train_examples :", info.splits['train'].num_examples, " num_test_examples: ",
 info.splits['test'].num_examples)

 train_dataset = mnist_train.map(scale).cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
 test_dataset = mnist_test.map(scale).batch(BATCH_SIZE)
 train = shard(train_dataset)
 test = shard(test_dataset)
 return train, test, info

def shard(dataset):
 options = tf.data.Options()
 options.experimental_distribute.auto_shard_policy = AutoShardPolicy.DATA
 return dataset.with_options(options)

```

(continues on next page)

(continued from previous page)

```

def decay(epoch):
 if epoch < 3:
 return 1e-3
 elif epoch >= 3 and epoch < 7:
 return 1e-4
 else:
 return 1e-5

def get_callbacks(model, checkpoint_dir="/opt/ml/checkpoints"):
 checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

 class PrintLR(tf.keras.callbacks.Callback):
 def on_epoch_end(self, epoch, logs=None):
 print('\nLearning rate for epoch {} is {}'.format(epoch + 1, model.optimizer.
↪lr.numpy()), flush=True)

 callbacks = [
 tf.keras.callbacks.TensorBoard(log_dir='./logs'),
 tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_prefix,
 # save_weights_only=True
),
 tf.keras.callbacks.LearningRateScheduler(decay),
 PrintLR()
]
 return callbacks

def build_and_compile_cnn_model():
 print("TF_CONFIG in model:", os.environ.get("TF_CONFIG"))
 model = tf.keras.Sequential([
 tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
 tf.keras.layers.MaxPooling2D(),
 tf.keras.layers.Flatten(),
 tf.keras.layers.Dense(64, activation='relu'),
 tf.keras.layers.Dense(10)
])

 model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
 optimizer=tf.keras.optimizers.Adam(),
 metrics=['accuracy'])

 return model

```

And, save the following script as *train.py*

```

import tensorflow as tf
import argparse
import mnist

print(tf.__version__)

```

(continues on next page)

(continued from previous page)

```

parser = argparse.ArgumentParser(description='Tensorflow Native MNIST Example')
parser.add_argument('--data-dir',
 help='location of the training dataset in the local filesystem (will_
↳ be downloaded if needed)',
 default='/code/data')
parser.add_argument('--data-bckt',
 help='location of the training dataset in an object storage bucket',
 default=None)

args = parser.parse_args()

artifacts_dir, checkpoint_dir, model_dir = mnist.create_dirs()

strategy = tf.distribute.MirroredStrategy()
print('Number of devices: {}'.format(strategy.num_replicas_in_sync))

train_dataset, test_dataset, info = mnist.get_data(data_bckt=args.data_bckt, data_
↳ dir=args.data_dir,
 num_replicas=strategy.num_replicas_in_
↳ sync)
with strategy.scope():
 model = mnist.build_and_compile_cnn_model()

model.fit(train_dataset, epochs=2, callbacks=mnist.get_callbacks(model, checkpoint_dir))

model.save(model_dir, save_format='tf')

```

**Initialize a distributed-training folder:**

At this point you have created a training file (or files) - train.py from the above example. Now, run the command below.

```
ads opctl distributed-training init --framework tensorflow --version v1
```

This will download the tensorflow framework and place it inside 'oci\_dist\_training\_artifacts' folder.

**Note:** Whenever you change the code, you have to build, tag and push the image to repo. This is automatically done in `ads opctl run` cli command.

**Containerize your code and build container:**

The required python dependencies are provided inside the conda environment file *oci\_dist\_training\_artifacts/tensorflow/v1/environments.yaml*. If your code requires additional dependency, update this file.

Also, while updating *environments.yaml* do not remove the existing libraries. You can append to the list.

Update the TAG and the IMAGE\_NAME as per your needs -

```

export IMAGE_NAME=<region.ocir.io/my-tenancy/image-name>
export TAG=latest
export MOUNT_FOLDER_PATH=.

```

Build the container image.

```
ads opctl distributed-training build-image \
 -t $TAG \
 -reg $IMAGE_NAME \
 -df oci_dist_training_artifacts/tensorflow/v1/Dockerfile \
```

The code is assumed to be in the current working directory. To override the source code directory, use the `-s` flag and specify the code dir. This folder should be within the current working directory.

```
ads opctl distributed-training build-image \
 -t $TAG \
 -reg $IMAGE_NAME \
 -df oci_dist_training_artifacts/tensorflow/v1/Dockerfile \
 -s $MOUNT_FOLDER_PATH
```

If you are behind proxy, ads opctl will automatically use your proxy settings (defined via `no_proxy`, `http_proxy` and `https_proxy`).

### Define your workload yaml:

The yaml file is a declarative way to express the workload. In this example, we bring up 1 worker node and 1 chief-worker node. The training code to run is `train.py`. All your training code is assumed to be present inside `/code` directory within the container. Additionally, you can also put any data files inside the same directory (and pass on the location ex `/code/data/**` as an argument to your training script using `runtime->spec->args`).

```
kind: distributed
apiVersion: v1.0
spec:
 infrastructure:
 kind: infrastructure
 type: dataScienceJob
 apiVersion: v1.0
 spec:
 projectId: oci.xxxx.<project_ocid>
 compartmentId: oci.xxxx.<compartment_ocid>
 displayName: Tensorflow
 logGroupId: oci.xxxx.<log_group_ocid>
 subnetId: oci.xxxx.<subnet-ocid>
 shapeName: VM.GPU2.1
 blockStorageSize: 50
 cluster:
 kind: TENSORFLOW
 apiVersion: v1.0
 spec:
 image: "@image"
 workDir: "oci://<bucket_name>@<bucket_namespace>/<bucket_prefix>"
 name: "tf_multiworker"
 config:
 env:
 - name: WORKER_PORT #Optional. Defaults to 12345
 value: 12345
 - name: SYNC_ARTIFACTS #Mandatory: Switched on by Default.
 value: 1
 - name: WORKSPACE #Mandatory if SYNC_ARTIFACTS==1: Destination object bucket.
 value: "oci://<bucket_name>@<bucket_namespace>/<bucket_prefix>"
 ↪ to sync generated artifacts to.
```

(continues on next page)



(continued from previous page)

```

 value: "<bucket_name>"
 - name: WORKSPACE_PREFIX #Mandatory if SYNC_ARTIFACTS==1: Destination object.
↪ bucket folder to sync generated artifacts to.
 value: "<bucket_prefix>"
 main:
 name: "chief"
 replicas: 1 #this will be always 1.
 worker:
 name: "worker"
 replicas: 1 #number of workers. This is in addition to the 'chief' worker. Could
↪ be more than 1
 runtime:
 kind: python
 apiVersion: v1.0
 spec:
 entryPoint: "/code/train.py" #location of user's training script in the container.
↪ image.
 args: #any arguments that the training script requires.
 - --data-dir # assuming data folder has been bundled in the container image.
 - /code/data/
 env:

```

Use `ads opctl` to create the cluster infrastructure and run the workload:

Do a dry run to inspect how the yaml translates to Job and Job Runs

```
ads opctl run -f train.yaml --dry-run
```

This will give output similar to this.

```

-----Entering dryrun mode-----
Creating Job with payload:
kind: job
spec:
 infrastructure:
 kind: infrastructure
 spec:
 projectId: oci.xxxx.<project_ocid>
 compartmentId: oci.xxxx.<compartment_ocid>
 displayName: Tensorflow
 logGroupId: oci.xxxx.<log_group_ocid>
 logId: oci.xxx.<log_ocid>
 subnetId: oci.xxxx.<subnet-ocid>
 shapeName: VM.GPU2.1
 blockStorageSize: 50
 type: dataScienceJob
 name: tf_multiworker
 runtime:
 kind: runtime
 spec:
 entrypoint: null
 env:
 - name: WORKER_PORT

```

(continues on next page)

(continued from previous page)

```

 value: 12345
 - name: SYNC_ARTIFACTS
 value: 1
 - name: WORKSPACE
 value: "<bucket_name>"
 - name: WORKSPACE_PREFIX
 value: "<bucket_prefix>"
 - name: OCI__WORK_DIR
 value: oci://<bucket_name>@<bucket_namespace>/<bucket_prefix>
 - name: OCI__EPHEMERAL
 value: None
 - name: OCI__CLUSTER_TYPE
 value: TENSORFLOW
 - name: OCI__WORKER_COUNT
 value: '1'
 - name: OCI__START_ARGS
 value: ''
 - name: OCI__ENTRY_SCRIPT
 value: /code/train.py
 image: "<region>.ocir.io/<tenancy_id>/<repo_name>/<image_name>:<image_tag>"
 type: container

```

```

+++++
Creating Main Job Run with following details:
Name: chief
Environment Variables:
 OCI__MODE:MAIN
~~~~~
Creating Job Runs with following details:
Name: worker_0
Environment Variables:
  OCI__MODE:WORKER
-----Ending dryrun mode-----

```

**Test Locally:**

Before submitting the workload to jobs, you can run it locally to test your code, dependencies, configurations etc. With `-b local` flag, it uses a local backend. Further when you need to run this workload on odsc jobs, simply use `-b job` flag instead.

```
ads opctl run -f train.yaml -b local
```

If your code requires to use any oci services (like object bucket), you need to mount oci keys from your local host machine onto the container. This is already done for you assuming the typical location of oci keys `~/oci`. You can modify it though, in-case you have keys at a different location. You need to do this in the config.ini file.

```
oci_key_mnt = ~/oci:/home/oci_dist_training/.oci
```

**Submit the workload:**

```
ads opctl run -f train.yaml -b job
```

**Note::** This will automatically push the docker image to the OCI [container registry repo](#) .

Once running, you will see on the terminal an output similar to the below. Note that this yaml can be used as input to `ads opctl distributed-training show-config -f <info.yaml>` - to both save and see the run info use `tee` - for example:

```
ads opctl run -f train.yaml | tee info.yaml
```

Listing 11: info.yaml

```
jobId: oci.xxxx.<job_ocid>
mainJobRunId:
  mainJobRunIdName: oci.xxxx.<job_run_ocid>
workDir: oci://my-bucket@my-namespace/daskcluster-testing/005
otherJobRunIds:
  - workerJobRunIdName_1: oci.xxxx.<job_run_ocid>
  - workerJobRunIdName_2: oci.xxxx.<job_run_ocid>
  - workerJobRunIdName_3: oci.xxxx.<job_run_ocid>
```

### Saving Artifacts to Object Storage Buckets

In case you want to save the artifacts generated by the training process (model checkpoints, TensorBoard logs, etc.) to an object bucket you can use the 'sync' feature. The environment variable `OCI__SYNC_DIR` exposes the directory location that will be automatically synchronized to the configured object storage bucket location. Use this directory in your training script to save the artifacts.

To configure the destination object storage bucket location, use the following settings in the workload yaml file(train.yaml).

```
- name: SYNC_ARTIFACTS
  value: 1
- name: WORKSPACE
  value: "<bucket_name>"
- name: WORKSPACE_PREFIX
  value: "<bucket_prefix>"
```

**Note:** Change `SYNC_ARTIFACTS` to `0` to disable this feature. Use `OCI__SYNC_DIR` env variable in your code to save the artifacts. For Example :

```
tf.keras.callbacks.ModelCheckpoint(os.path.join(os.environ.get("OCI__SYNC_DIR"), "ckpts",
↪ 'checkpoint-{{epoch}}.h5'))
```

### Monitoring the workload logs

To view the logs from a job run, you could run -

```
ads jobs watch oci.xxxx.<job_run_ocid>
```

### Profiling

You may want to profile your training setup for optimization/performance tuning. Profiling typically provides a detailed analysis of cpu utilization, gpu utilization, top cuda kernels, top operators etc. You can choose to profile your training setup using the native Pytorch profiler or using a third party profiler such as [Nvidia Nsights](#).

#### Profiling using Tensorflow Profiler

[Tensorflow Profiler](#) is a native offering from Tensorflow for Tensorflow performance profiling.

Profiling is invoked using code instrumentation using one of the following apis.

```
tf.keras.callbacks.TensorBoard
```

```
tf.profiler.experimental.Profile
```

Refer above links for changes that you need to do in your training script for instrumentation.

You should choose the OCI\_\_SYNC\_DIR directory to save the profiling logs. For example:

```
options = tf.profiler.experimental.ProfilerOptions(
    host_tracer_level=2,
    python_tracer_level=1,
    device_tracer_level=1,
    delay_ms=None)
with tf.profiler.experimental.Profile(os.environ.get("OCI__SYNC_DIR") + "/logs",
    ↪options=options):
    # training code
```

In case of keras callback:

```
tboard_callback = tf.keras.callbacks.TensorBoard(log_dir = os.environ.get("OCI__SYNC_DIR"
    ↪) + "/logs",
                                                    histogram_freq = 1,
                                                    profile_batch = '500,520')
model.fit(...,callbacks = [tboard_callback])
```

Also, the sync feature SYNC\_ARTIFACTS should be enabled '1' to sync the profiling logs to the configured object storage.

Thereafter, use Tensorboard to view logs. Refer the [Tensorboard setup](#) for set-up on your computer.

### Profiling using Nvidia Nsights

[Nvidia Nsights](#). is a system wide profiling tool from Nvidia that can be used to profile Deep Learning workloads.

Nsights requires no change in your training code. This works on process level. You can enable this experimental feature in your training setup via the following configuration in the runtime yaml file(highlighted).

```
spec:
  image: "@image"
  workDir: "oci://@"
  name: "tf_multiworker"
  config:
    env:
      - name: WORKER_PORT
        value: 12345
      - name: SYNC_ARTIFACTS
        value: 1
      - name: WORKSPACE
        value: "<bucket_name>"
      - name: WORKSPACE_PREFIX
        value: "<bucket_prefix>"
      - name: PROFILE
        value: 1
      - name: PROFILE_CMD
        value: "nsys profile -w true -t cuda,nvtx,osrt,cudnn,cublas -s none -o /opt/
    ↪ml/nsight_report -x true"
  main:
```

(continues on next page)

(continued from previous page)

```

    name: "main"
    replicas: 1
  worker:
    name: "worker"
    replicas: 1

```

Refer [this](#) for nsys profile command options. You can modify the command within the PROFILE\_CMD but remember this is all experimental. The profiling reports are generated per node. You need to download the reports to your computer manually or via the oci command.

```

oci os object bulk-download \
  -ns <namespace> \
  -bn <bucket_name> \
  --download-dir /path/on/your/computer \
  --prefix path/on/bucket/<job_id>

```

**Note:** -bn == WORKSPACE and --prefix path == WORKSPACE\_PREFIX/<job\_id> , as configured in the runtime yaml file. To view the reports, you would need to install Nsight Systems app from [here](#). Thereafter, open the downloaded reports in the Nsight Systems app.

### Other Tensorflow Strategies supported

Tensorflow has two multi-worker strategies: MultiWorkerMirroredStrategy and ParameterServerStrategy. Let's see changes that you would need to do to run ParameterServerStrategy workload.

You can have the following training Tensorflow script for ParameterServerStrategy saved as train.py (just like mnist.py and train.py in case of MultiWorkerMirroredStrategy):

```

# Script adapted from tensorflow tutorial: https://www.tensorflow.org/tutorials/
↪distribute/parameter_server_training

import os
import tensorflow as tf
import json
import multiprocessing

NUM_PS = len(json.loads(os.environ['TF_CONFIG'])['cluster']['ps'])
global_batch_size = 64

def worker(num_workers, cluster_resolver):
    # Workers need some inter_ops threads to work properly.
    worker_config = tf.compat.v1.ConfigProto()
    if multiprocessing.cpu_count() < num_workers + 1:
        worker_config.inter_op_parallelism_threads = num_workers + 1

    for i in range(num_workers):
        print("cluster_resolver.task_id: ", cluster_resolver.task_id, flush=True)

        s = tf.distribute.Server(
            cluster_resolver.cluster_spec(),
            job_name=cluster_resolver.task_type,
            task_index=cluster_resolver.task_id,
            config=worker_config,

```

(continues on next page)

(continued from previous page)

```

        protocol="grpc")
    s.join()

def ps(num_ps, cluster_resolver):
    print("cluster_resolver.task_id: ", cluster_resolver.task_id, flush=True)
    for i in range(num_ps):
        s = tf.distribute.Server(
            cluster_resolver.cluster_spec(),
            job_name=cluster_resolver.task_type,
            task_index=cluster_resolver.task_id,
            protocol="grpc")
        s.join()

def create_cluster(cluster_resolver, num_workers=1, num_ps=1, mode="worker"):
    os.environ["GRPC_FAIL_FAST"] = "use_caller"

    if mode.lower() == 'worker':
        print("Starting worker server...", flush=True)
        worker(num_workers, cluster_resolver)
    else:
        print("Starting ps server...", flush=True)
        ps(num_ps, cluster_resolver)

    return cluster_resolver, cluster_resolver.cluster_spec()

def decay(epoch):
    if epoch < 3:
        return 1e-3
    elif epoch >= 3 and epoch < 7:
        return 1e-4
    else:
        return 1e-5

def get_callbacks(model):
    class PrintLR(tf.keras.callbacks.Callback):
        def on_epoch_end(self, epoch, logs=None):
            print('\nLearning rate for epoch {} is {}'.format(epoch + 1, model.optimizer.
↪lr.numpy()), flush=True)

    callbacks = [
        tf.keras.callbacks.TensorBoard(log_dir='./logs'),
        tf.keras.callbacks.LearningRateScheduler(decay),
        PrintLR()
    ]
    return callbacks

def create_dir(dir):
    if not os.path.exists(dir):
        os.makedirs(dir)

```

(continues on next page)

(continued from previous page)

```

def get_artificial_data():
    x = tf.random.uniform((10, 10))
    y = tf.random.uniform((10,))

    dataset = tf.data.Dataset.from_tensor_slices((x, y)).shuffle(10).repeat()
    dataset = dataset.batch(global_batch_size)
    dataset = dataset.prefetch(2)
    return dataset

cluster_resolver = tf.distribute.cluster_resolver.TFConfigClusterResolver()
if not os.environ["OCI__MODE"] == "MAIN":
    create_cluster(cluster_resolver, num_workers=1, num_ps=1, mode=os.environ["OCI__MODE"]
    ↪")
    pass

variable_partitioner = (
    tf.distribute.experimental.partitioners.MinSizePartitioner(
        min_shard_bytes=(256 << 10),
        max_shards=NUM_PS))

strategy = tf.distribute.ParameterServerStrategy(
    cluster_resolver,
    variable_partitioner=variable_partitioner)

dataset = get_artificial_data()

with strategy.scope():
    model = tf.keras.models.Sequential([tf.keras.layers.Dense(10)])
    model.compile(tf.keras.optimizers.SGD(), loss="mse", steps_per_execution=10)

callbacks = get_callbacks(model)
model.fit(dataset, epochs=5, steps_per_epoch=20, callbacks=callbacks)

```

Train.yaml: The only difference here is that the parameter server train.yaml also needs to have ps worker-pool. This will create dedicated instance(s) for Tensorflow Parameter Servers.

Use the following train.yaml:

```

kind: distributed
apiVersion: v1.0
spec:
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    apiVersion: v1.0
    spec:
      projectId: oci.xxxx.<project_ocid>
      compartmentId: oci.xxxx.<compartment_ocid>
      displayName: Distributed-TF
      logGroupId: oci.xxxx.<log_group_ocid>

```

(continues on next page)

(continued from previous page)

```

    subnetId: oci.xxxx.<subnet-ocid>
    shapeName: VM.Standard2.4
    blockStorageSize: 50
cluster:
  kind: TENSORFLOW
  apiVersion: v1.0
  spec:
    image: "@image"
    workDir: "oci://<bucket_name>@<bucket_namespace>/<bucket_prefix>"
    name: "tf_ps"
    config:
      env:
        - name: WORKER_PORT #Optional. Defaults to 12345
          value: 12345
        - name: SYNC_ARTIFACTS #Mandatory: Switched on by Default.
          value: 1
        - name: WORKSPACE #Mandatory if SYNC_ARTIFACTS==1: Destination object bucket.
          value: "<bucket_name>"
        - name: WORKSPACE_PREFIX #Mandatory if SYNC_ARTIFACTS==1: Destination object.
          value: "<bucket_prefix>"
      main:
        name: "coordinator"
        replicas: 1 #this will be always 1.
      worker:
        name: "worker"
        replicas: 1 #number of workers; any number > 0
      ps:
        name: "ps" # number of parameter servers; any number > 0
        replicas: 1
  runtime:
    kind: python
    apiVersion: v1.0
    spec:
      entryPoint: "/code/train.py" #location of user's training script in the container.
      args: #any arguments that the training script requires.
      env:

```

The rest of the steps remain the same and should be followed as it is.



## 11.2.8 Run Source Code from Git or Object Storage

### Require ADS >=2.6.3

Running source code from Git or Object Storage requires ADS 2.6.3 or newer.

```
python3 -m pip install oracle-ads>=2.6.3 --upgrade
```

Instead of adding the training source code to the docker image, you can also fetch the code at runtime from Git repository or object storage.

### 11.2.8.1 Git Repository

To fetch code from Git repository, you can update the runtime section of the yaml to specify `type` as `git` and add `uri` of the Git repository to the `runtime.spec` section. For example:

```
1 runtime:
2   apiVersion: v1
3   kind: python
4   type: git
5   spec:
6     uri: git@github.com:username/repository.git
7     branch: develop
8     commit: abcdef
9     gitSecretId: ocid1.xxxxxx
10    entryPoint: "train.py"
```

The spec supports the following options:

- `uri`, the URI of the git repository. This can be `http` or `https` URI for public repository. For private repository, please use `ssh` or `git@` URI.
- `branch`, the Git branch. The default branch (usually `main`) will be used if this is not specified.
- `commit`, the Git commit. The latest commit will be used if this is not specified.
- `gitSecretId`, the OCID of secret from OCI vault, which stores the SSH key for accessing private repository.
- `entryPoint`, the file path to start the training code. The can be the relative path relative to the root of the git repository. The source code is cloned to the `/code` directory. You may also use the absolute path.

To clone the git repository, your subnet needs egress from port 80 for `http`, 443 for `https`, or 22 for `ssh`.

You can config proxy for git clone by setting the corresponding `ssh_proxy`, `http_proxy` or `https_proxy` environment variable to the proxy address. If you configured `https_proxy` or `http_proxy`, you also need to add all IP addresses in your subnet to the `no_proxy` environment variable since communications between training nodes should not go through proxy.

### 11.2.8.2 Object Storage

To fetch code from Object Storage, you can update the `runtime` section of the `yaml` to specify `type` as `remote` and add `uri` of the OCI object storage to the `runtime.spec` section. For example:

```

1 runtime:
2   apiVersion: v1
3   kind: python
4   type: remote
5   spec:
6     uri: oci://bucket@namespace/prefix/to/source_code_dir
7     entryPoint: "/code/source_code_dir/train.py"

```

The `uri` can be a single file or a prefix (directory). The `entryPoint` is the the file path to start the training code. When using relative path, if `uri` is a single file, `entryPoint` should be the filename. If `uri` is a directory, the `entryPoint` should contain the name of the directory like the example above. The source code is cloned to the `/code` directory. You may also use the absolute path.

### 11.2.9 YAML Schema

The distributed training workload is defined in `YAML` and can be launched by invoking the `ads opctl run -f path/to/yaml` command.

Following is the `YAML` schema for validating the `YAML` using [Cerberus](#):

```

1 kind:
2   type: string
3   allowed:
4     - distributed
5   apiVersion:
6     type: string
7   spec:
8     type: dict
9     schema:
10      infrastructure:
11        type: dict
12        schema:
13          kind:
14            type: string
15            allowed:
16              - infrastructure
17          type:
18            type: string
19            allowed:
20              - dataScienceJob
21          apiVersion:
22            type: string
23          spec:
24            type: dict

```

(continues on next page)

(continued from previous page)

```

25     schema:
26         displayName:
27             type: string
28         compartmentId:
29             type: string
30         projectId:
31             type: string
32         logGroupId:
33             type: string
34         logId:
35             type: string
36         subnetId:
37             type: string
38         shapeName:
39             type: string
40         blockStorageSize:
41             type: integer
42             min: 50
43 cluster:
44     type: dict
45     schema:
46         kind:
47             type: string
48         allowed:
49             - PYTORCH
50             - DASK
51             - HOROVOD
52             - dask
53             - pytorch
54             - horovod
55         apiVersion:
56             type: string
57         spec:
58             type: dict
59             schema:
60                 image:
61                     type: string
62                 workDir:
63                     type: string
64                 name:
65                     type: string
66                 config:
67                     type: dict
68                     nullable: true
69                     schema:
70                         startOptions:
71                             type: list
72                             schema:
73                                 type: string
74                         env:
75                             type: list
76                             nullable: true

```

(continues on next page)

(continued from previous page)

```
77     schema:
78         type: dict
79         schema:
80             name:
81                 type: string
82             value:
83                 type:
84                 - number
85                 - string
86 main:
87     type: dict
88     schema:
89         name:
90             type: string
91     replicas:
92         type: integer
93     config:
94         type: dict
95         nullable: true
96         schema:
97             env:
98                 type: list
99                 nullable: true
100             schema:
101                 type: dict
102                 schema:
103                     name:
104                         type: string
105                     value:
106                         type:
107                         - number
108                         - string
109 worker:
110     type: dict
111     schema:
112         name:
113             type: string
114     replicas:
115         type: integer
116     config:
117         type: dict
118         nullable: true
119         schema:
120             env:
121                 type: list
122                 nullable: true
123             schema:
124                 type: dict
125                 schema:
126                     name:
127                         type: string
128                     value:
```

(continues on next page)

(continued from previous page)

```

129         type:
130         - number
131         - string
132     runtime:
133     type: dict
134     schema:
135     kind:
136     type: string
137     apiVersion:
138     type: string
139     spec:
140     type: dict
141     schema:
142     entryPoint:
143     type: string
144     kwargs:
145     type: string
146     args:
147     type: list
148     schema:
149     type:
150     - number
151     - string
152     env:
153     type: list
154     nullable: true
155     schema:
156     type: dict
157     schema:
158     name:
159     type: string
160     value:
161     type:
162     - number
163     - string

```

### 11.2.10 Troubleshooting

Check guide for troubleshooting and known issues.

## 11.3 TensorBoard

TensorBoard helps visualizing your experiments. You bring up a TensorBoard session on your workstation and point to the directory that contains the TensorBoard logs.

### Prerequisite

1. Object storage bucket
2. Access to Object Storage bucket from your workstation
3. ocifs version 1.1.0 and above

### 11.3.1 Setting up local environment

It is required that `tensorboard` is installed in a dedicated conda environment or virtual environment. Prepare an environment yaml file for creating conda environment with following command -

```
cat <<EOF > tensorboard-dep.yaml
dependencies:
- python=3.8
- pip
- pip:
  - ocifs
  - tensorboard
name: tensorboard
EOF
```

Create the conda environment from the yaml file generated in the preceeding step

```
conda env create -f tensorboard-dep.yaml
```

This will create a conda environment called `tensorboard`. Activate the conda environment by running -

```
conda activate tensorboard
```

### 11.3.2 Viewing logs from your experiments

To launch a TensorBoard session on your local workstation, run -

```
export OCIFS_IAM_KEY=api_key # If you are using resource principal, set resource_
↪principal
tensorboard --logdir oci://my-bucket@my-namespace/path/to/logs
```

This will bring up TensorBoard app on your workstation. Access TensorBoard at `http://localhost:6006/`

**Note:** The logs take some initial time (few minutes) to reflect on the tensorboard dashboard.

### 11.3.3 Writing TensorBoard logs to Object Storage

#### Prerequisite

1. `tensorboard` is installed.
2. `ocifs` version is 1.1.0 and above.
3. `oracle-ads` version 2.6.0 and above.

### 11.3.3.1 PyTorch

You could write your logs from your PyTorch experiments directly to object storage and view the logs on TensorBoard running on your local workstation in real time. Here is an example of running PyTorch experiment and writing TensorBoard logs from OCI Data Science Notebook

1. Create or Open an existing OCI Data Science Notebook session
2. Run `odsc conda install -s pytorch110_p37_cpu_v1` on terminal inside the notebook session
3. Activate conda environment - `conda activate /home/datascience/conda/pytorch110_p37_cpu_v1`
4. Install TensorBoard - `python3 -m pip install tensorboard`
5. Upgrade to latest ocifs - `python3 -m pip install ocifs --upgrade`
6. Create a notebook and select `pytorch110_p37_cpu_v1` kernel
7. Copy the following code into a cell and update the object storage path in the code snippet

```
# Reference: https://github.com/pytorch/tutorials/blob/master/recipes_source/recipes/
↪ tensorboard_with_pytorch.py

import torch
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter("oci://my-bucket@my-namespace/path/to/logs")

x = torch.arange(-5, 5, 0.1).view(-1, 1)
y = -5 * x + 0.1 * torch.randn(x.size())

model = torch.nn.Linear(1, 1)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.1)

def train_model(iter):
    for epoch in range(iter):
        y1 = model(x)
        loss = criterion(y1, y)
        writer.add_scalar("Loss/train", loss, epoch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

train_model(10)
writer.flush()
writer.close()
```

7. Run the cell
8. View the logs from your workstation while the experiment is in progress by launching TensorBoard with following command -

```
OCIFS_IAM_TYPE=api_key tensorboard --logdir "oci://my-bucket@my-namespace/path/to/logs"
```

For more possibilities with TensorBoard and PyTorch check this [link](#)

### 11.3.3.2 TensorFlow

Currently TensorFlow cannot write directly to object storage. However, we can create logs in the local directory and then copy the logs over to object storage, which then can be viewed from the TensorBoard running on your local workstation.

When you run a OCI Data Science Job with `ads.jobs.NotebookRuntime` or `ads.jobs.GitRuntime`, all the output is automatically copied over to the configured object storage bucket.

#### 11.3.3.2.1 OCI Data Science Notebook

Here is an example of running a TensorFlow experiment in OCI Data Science Notebook and then viewing the logs from TensorBoard

1. Create or open an existing notebook session.
2. Download notebook - [https://raw.githubusercontent.com/mayoor/stats-ml-exps/master/tensorboard\\_tf.ipynb](https://raw.githubusercontent.com/mayoor/stats-ml-exps/master/tensorboard_tf.ipynb)

```
!wget https://raw.githubusercontent.com/mayoor/stats-ml-exps/master/tensorboard_tf.ipynb
```

3. Run `odsc conda install -s tensorflow27_p37_cpu_v1` on terminal to install TensorFlow 2.6 environment.
4. Open the downloaded notebook - `tensorboard_tf.ipynb`
5. Select `tensorflow27_p37_cpu_v1` kernel.
6. Run all cells.
7. Copy TensorBoard logs folder - `tflogs` to object storage using `oci-cli`

```
oci os object bulk-upload -bn "<my-bucket>" -ns "<my-namespace>" --src-dir tflogs --
↪prefix myexperiment/tflogs/
```

View the logs from you workstation once the logs are uploaded by launching the TensorBoard with following command -

```
OCIFS_IAM_TYPE=api_key tensorboard --logdir "oci://my-bucket@my-namespace/myexperiment/
↪tflogs/"
```

#### 11.3.3.2.2 OCI Data Science Jobs

Here is an example of running a TensorFlow experiment in OCI Data Science Jobs and then viewing the logs from TensorBoard

1. Run the following code to submit a notebook to OCI Data Science Job. You could run this code snippet from your local workstation or OCI Data Science Notebook session. You need `oracle-ads` version `>= 2.6.0`.

```
from ads.jobs import Job, DataScienceJob, NotebookRuntime
# Define an OCI Data Science job to run a jupyter Python notebook
job = (
    Job(name="<job_name>")
    .with_infrastructure(
        # The same configurations as the OCI notebook session will be used.
        DataScienceJob()
    )
)
```

(continues on next page)



(continued from previous page)

```

        .with_log_group_id("oci.xxxx.<log_group_ocid>")
        .with_log_id("oci.xxx.<log_ocid>")
        .with_project_id("oci.xxxx.<project_ocid>")
        .with_shape_name("VM.Standard2.1")
        .with_subnet_id("oci.xxxx.<subnet-ocid>")
        .with_block_storage_size(50)
        .with_compartment_id("oci.xxxx.<compartment_ocid>")
    )
    .with_runtime(
        NotebookRuntime()
        .with_notebook("https://raw.githubusercontent.com/mayoor/stats-ml-exps/master/
↪ tensorboard_tf.ipynb")
        .with_service_conda("tensorflow27_p37_cpu_v1")
        # Saves the notebook with outputs to OCI object storage.
        .with_output("oci://my-bucket@my-namespace/myexperiment/jobs/")
    )
).create()
# Run and monitor the job
run = job.run().watch()

```

View the logs from your workstation once the job is complete by launching the tensorboard with following command -

```

OCIFS_IAM_TYPE=api_key tensorboard --logdir "oci://my-bucket@my-namespace//myexperiment/
↪ jobs/tflogs/"

```

## 11.4 Model Evaluation

With the ever-growing suite of models at the disposal of data scientists, the problems with selecting a model have grown similarly. ADS offers the Evaluation Class, a collection of tools, metrics, and charts concerned with the contradiction of several models.

After working hard to architect and train your model, it's important to understand how it performs across a series of benchmarks. Evaluation is a set of functions that convert the output of your test data into an interpretable, standardized series of scores and charts. From the accuracy of the ROC curve and residual QQ plots.

Evaluation can help machine learning developers to:

- Quickly compare models across several industry-standard metrics.
  - For example, what's the accuracy, and F1-Score of my binary classification model?
- Discover where a model is failing to feed back into future model development.
  - For example, while accuracy is high, precision is low, which is why the examples I care about are failing.
- Increase understanding of the trade-offs of various model types.

Evaluation helps you understand where the model is likely to perform well or not. For example, model A performs well when the weather is clear, but is much more uncertain during inclement conditions.

There are three types of ADS Evaluators, binary classifier, multinomial classifier, and regression.

## 11.4.1 Quick Start

### 11.4.1.1 Comparing Binary Classification Models

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from ads.common.model import ADSModel
from ads.common.data import ADSData
from ads.evaluations.evaluator import ADSEvaluator

seed = 42

X, y = make_classification(n_samples=10000, n_features=25, n_classes=2, flip_y=0.1)

trainx, testx, trainy, testy = train_test_split(X, y, test_size=0.30, random_state=seed)

lr_clf = LogisticRegression(
    random_state=0, solver="lbfgs", multi_class="multinomial"
).fit(trainx, trainy)

rf_clf = RandomForestClassifier(n_estimators=50).fit(trainx, trainy)

bin_lr_model = ADSModel.from_estimator(lr_clf, classes=[0, 1])
bin_rf_model = ADSModel.from_estimator(rf_clf, classes=[0, 1])

evaluator = ADSEvaluator(
    ADSData(testx, testy),
    models=[bin_lr_model, bin_rf_model],
    training_data=ADSData(trainx, trainy),
)

print(evaluator.metrics)
```

### 11.4.1.2 Comparing Multi Classification Models

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from ads.common.model import ADSModel
from ads.common.data import ADSData
from ads.evaluations.evaluator import ADSEvaluator
```

(continues on next page)

(continued from previous page)

```

seed = 42

X, y = make_classification(
    n_samples=10000, n_features=25, n_classes=3, flip_y=0.1, n_clusters_per_class=1
)

trainx, testx, trainy, testy = train_test_split(X, y, test_size=0.30, random_state=seed)

lr_multi_clf = LogisticRegression(
    random_state=0, solver="lbfgs", multi_class="multinomial"
).fit(trainx, trainy)

rf_multi_clf = RandomForestClassifier(n_estimators=10).fit(trainx, trainy)

multi_lr_model = ADSModel.from_estimator(lr_multi_clf)
multi_rf_model = ADSModel.from_estimator(rf_multi_clf)

evaluator = ADSEvaluator(
    ADSData(testx, testy),
    models=[multi_lr_model, multi_rf_model],
)

print(evaluator.metrics)

```

#### 11.4.1.3 Comparing Regression Models

```

from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestClassifier

from ads.common.model import ADSModel
from ads.common.data import ADSData
from ads.evaluations.evaluator import ADSEvaluator

seed = 42

X, y = make_regression(n_samples=10000, n_features=10, n_informative=2, random_state=42)

trainx, testx, trainy, testy = train_test_split(X, y, test_size=0.3, random_state=seed)

lin_reg = LinearRegression().fit(trainx, trainy)

lasso_reg = Lasso(alpha=0.1).fit(trainx, trainy)

```

(continues on next page)

(continued from previous page)

```

lin_reg_model = ADSModel.from_estimator(lin_reg)
lasso_reg_model = ADSModel.from_estimator(lasso_reg)

reg_evaluator = ADSEvaluator(
    ADSData(testx, testy), models=[lin_reg_model, lasso_reg_model]
)

print(reg_evaluator.metrics)

```

## 11.4.2 Binary Classification

Binary classification is a type of modeling wherein the output is binary. For example, Yes or No, Up or Down, 1 or 0. These models are a special case of multinomial classification so have specifically catered metrics.

The prevailing metrics for evaluating a binary classification model are accuracy, hamming loss, kappa score, precision, recall,  $F_1$  and AUC. Most information about binary classification uses a few of these metrics to speak to the importance of the model.

- **Accuracy:** The proportion of predictions that were correct. It is generally converted to a percentage where 100% is a perfect classifier. An accuracy of 50% is random (for a balanced dataset) and an accuracy of 0% is a perfectly wrong classifier.
- **AUC:** Area Under the Curve (AUC) refers to the area under an ROC curve. This is a numerical way to summarize the robustness of a model to its discrimination threshold. The AUC is computed by integrating the area under the ROC curve. It is akin to the probability that your model scores better on results to which it accredits a higher score. Thus 1.0 is a perfect score, 0.5 is the average score of a random classifier, and 0.0 is a perfectly backward scoring classifier.
- **$F_1$  Score:** There is generally a trade-off between the precision and recall and the  $F_1$  score is a metric that combines them into a single number. The  $F_1$  Score is the harmonic mean of precision and recall:

$$F_1 = 2 \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Therefore a perfect  $F_1$  score is 1. That is, the classifier has perfect precision and recall. The worst  $F_1$  score is 0. The  $F_1$  score of a random classifier is heavily dependent on the nature of the data.

- **Hamming Loss:** The proportion of predictions that were incorrectly classified and is equivalent to  $1 - \text{accuracy}$ . This means a Hamming Loss of 0 is a perfect classifier. A score of 0.5 is a random classifier (for a balanced dataset), and 1 is a perfectly incorrect classifier.
- **Kappa Score:** Cohen's  $\kappa$  coefficient is a statistic that measures inter-annotator agreement. This function computes Cohen's  $\kappa$ , a score that expresses the level of agreement between two annotators on a classification problem. It is defined as:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

$p_o$  is the empirical probability of agreement on the label assigned to any sample (the observed agreement ratio).  $p_e$  is the expected agreement when both annotators assign labels randomly.  $p_e$  is estimated using a per-annotator empirical prior over the class labels.

- **Precision:** The proportion of the True class that were predicted to be True and are actually in the True class  $\frac{TP}{TP+FP}$ . This is also known as Positive Predictive Value (PPV). A precision of 1.0 is perfect precision, 0.0 is *bad* precision. However, the precision of a random classifier varies highly based on the nature of the data and to a lesser extent a *bad* precision.

- **Recall:** This is the proportion of the True class predictions that were correctly predicted over the number of True predictions (correct or incorrect)  $\frac{TP}{TP+FN}$ . This is also known as True Positive Rate (TPR) or Sensitivity. A recall of 1.0 is perfect recall, 0.0 is *bad* recall. however, the recall of a random classifier varies highly based on the nature of the data and to a lesser extent a *bad* recall.

The prevailing charts and plots for binary classification are the Precision-Recall Curve, the ROC curve, the Lift Chart, the Gain Chart, and the Confusion Matrix. These are inter-related with the previously described metrics and are commonly used in the binary classification literature.

- Confusion Matrix
- Gain Chart
- Lift Chart
- Precision-Recall Curve
- ROC curve

This code snippet demonstrates how to generate the above metrics and charts. The data has to be split into a testing and training set with the features in *X\_train* and *X\_test* and the responses in *y\_train* and *y\_test*.

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from ads.common.model import ADSModel
from ads.common.data import ADSData
from ads.evaluations.evaluator import ADSEvaluator

seed = 42

X, y = make_classification(n_samples=10000, n_features=25, n_classes=2, flip_y=0.1)

trainx, testx, trainy, testy = train_test_split(X, y, test_size=0.30, random_state=seed)

lr_clf = LogisticRegression(
    random_state=0, solver="lbfgs", multi_class="multinomial"
).fit(trainx, trainy)

rf_clf = RandomForestClassifier(n_estimators=50).fit(trainx, trainy)

bin_lr_model = ADSModel.from_estimator(lr_clf, classes=[0, 1])
bin_rf_model = ADSModel.from_estimator(rf_clf, classes=[0, 1])

evaluator = ADSEvaluator(
    ADSData(testx, testy),
    models=[bin_lr_model, bin_rf_model],
    training_data=ADSData(trainx, trainy),
)

print(evaluator.metrics)
```

To use the ADSEvaluator the standard sklearn models into ADSModels.

The ADSModel class in the ADS package has a `from_estimator` function that takes as input a fitted estimator and converts it into an ADSModel object. With classification, the class labels also need to be provided. The ADSModel object is used for evaluation by the ADSEvaluator object.

To show all of the metrics in a table, run:

```
evaluator.metrics
```

#### Evaluation Metrics (testing data):

	LogisticRegression	RandomForestClassifier
accuracy	0.9988	0.9991
hamming_loss	0.001156	0.0008839
kappa_score_	0.5915	0.7268
precision	0.9024	0.8814
recall	0.4405	0.619
f1	0.592	0.7273
auc	0.9245	0.9042

#### Evaluation Metrics (training data):

	LogisticRegression	RandomForestClassifier
accuracy	0.9989	0.9999
hamming_loss	0.001105	0.0001316
kappa_score_	0.583	0.9603
precision	0.8255	0.9926
recall	0.4512	0.9302
f1	0.5835	0.9604
auc	0.9164	1

Fig. 1: Evaluator Metrics

To show all of the charts, run:

```
evaluator.show_in_notebook(perfect=True)
```

Important parameters:

- If `perfect` is set to `True`, ADS plots a perfect classifier for comparison in Lift and Gain charts.
- If `baseline` is set to `True`, ADS won't include a baseline for the comparison of various plots.
- If `use_training_data` is set `True`, ADS plots the evaluations of the training data.
- If `plots` contain a list of plot types, ADS plots only those plot types.

This code snippet demonstrates how to add a custom metric, a  $F_2$  score, to the evaluator.

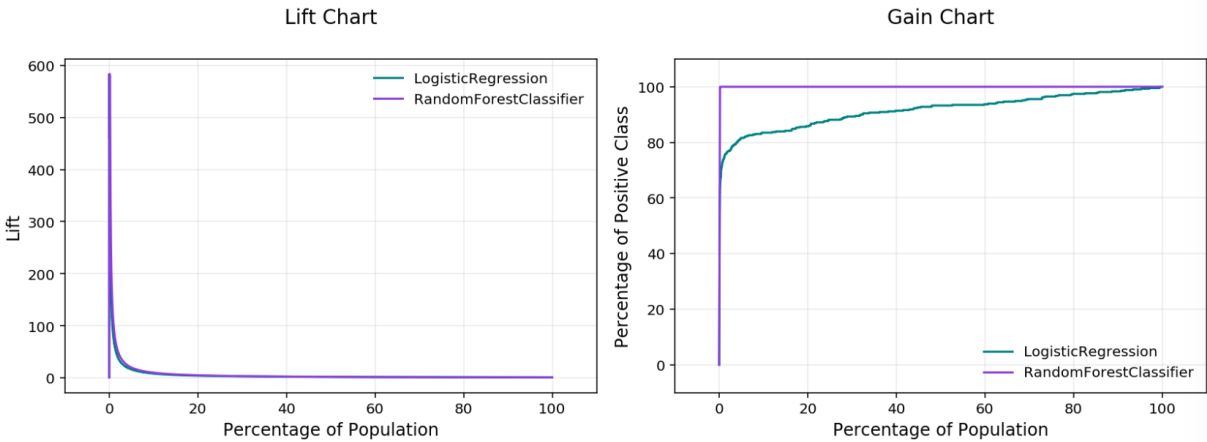


Fig. 2: Lift & Gain Chart

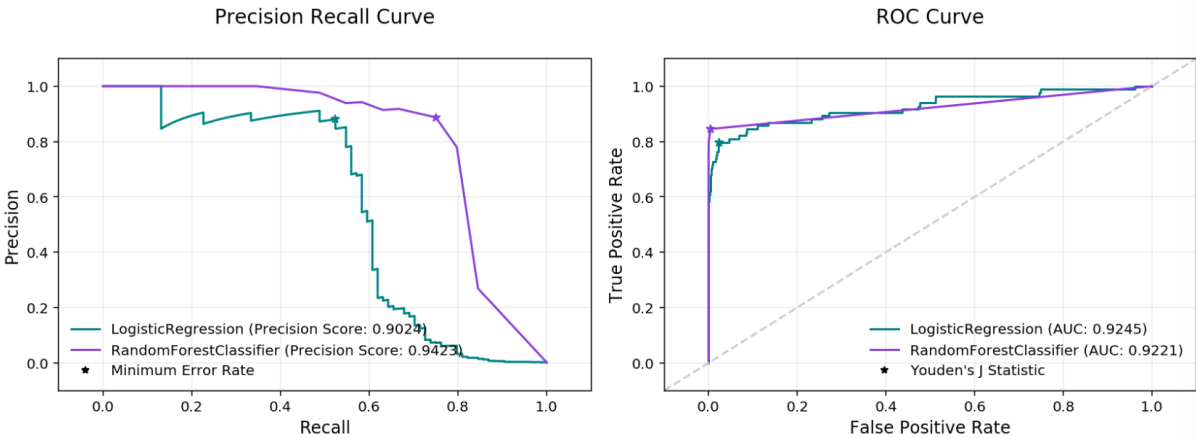


Fig. 3: Precision Recall and ROC Curves

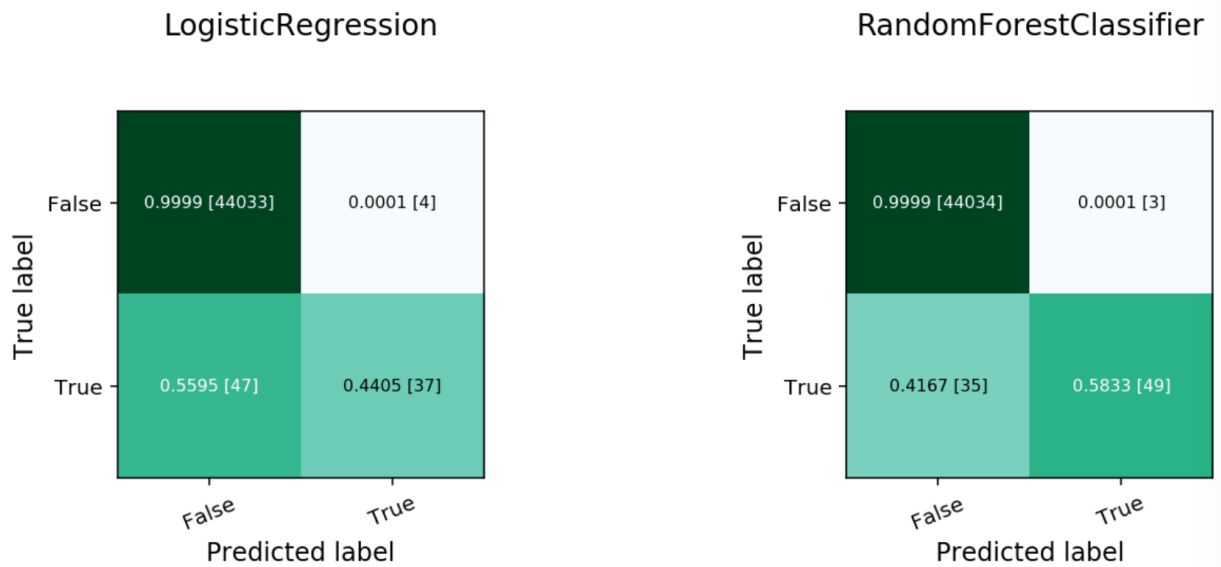


Fig. 4: Normalized Confusion Matrix

```

from ads.evaluations.evaluator import ADSEvaluator
evaluator = ADSEvaluator(test, models=[modelA, modelB, modelC modelD])

from sklearn.metrics import fbeta_score
def F2_Score(y_true, y_pred):
    return fbeta_score(y_true, y_pred, 2)
evaluator.add_metrics([F2_Score], ["F2 Score"])
evaluator.metrics

```

#### 11.4.2.1 Fairness Metrics

New in version 2.6.1..

Fairness metrics will be automatically generated for any feature specified in the `protected_features` argument to the `ADSEvaluator` object. The added metrics are:

- **Equal Odds:** For each of the *protected\_features* specified, Equal Odds is a ratio between the positive rates for each class within that feature. The closer this value is to 1, the less biased the model and data are with respect to the feature, F. In other terms, for a binary feature F with classes A and B, Equal Odds is calculated using the following formula:

$$\frac{P(\hat{y} = 1 | Y = y, F = A)}{P(\hat{y} = 1 | Y = y, F = B)}$$

- **Equal Opportunity:** For each of the *protected\_features* specified, Equal Opportunity is a ratio between the true positive rates for each class within that feature. The closer this value is to 1, the less biased the model is with respect to the feature F. In other terms, for a binary feature F with classes A and B, Equal Opportunity is



calculated using the following formula:

$$\frac{P(\hat{y} = 1|Y = 1, F = A)}{P(\hat{y} = 1|Y = 1, F = B)}$$

- **Statistical Parity:** For each of the *protected\_features* specified, Statistical Parity is a ratio between the prediction rates for each class within that feature. The closer this value is to 1, the less biased the model and data are with respect to the feature F. In other terms, for a binary feature F with classes A and B, Statistical Parity is calculated using the following formula:

$$\frac{P(\hat{y}|F = A)}{P(\hat{y}|F = B)}$$

The following plots are added to explain the fairness metrics:

- Equal Odds Bar Chart: False Positive Rate bar chart by protected feature class
- Equal Opportunity Bar Chart: True Positive Rate bar chart by protected feature class
- Statistical Parity Bar Chart: Number of positive predictions by protected feature class

If *protected\_features* contains a list of column names in *data.X*, ADS will generate fairness metrics for each of those columns.

### 11.4.3 Multinomial Classification

Multinomial classification is a type of modeling wherein the output is discrete. For example, an integer 1-10, an animal at the zoo, or a primary color. These models have a specialized set of charts and metrics for their evaluation.

The prevailing metrics for evaluating a multinomial classification model are:

- **Accuracy:** The proportion of predictions that were correct. It is generally converted to a percentage where 100% is a perfect classifier. For a balanced dataset, an accuracy of  $\frac{100\%}{k}$  where  $k$  is the number of classes, is a random classifier. An accuracy of 0% is a perfectly wrong classifier.
- **F<sub>1</sub> Score (weighted, macro or micro):** There is generally a trade-off between the precision and recall and the  $F_1$  score is a metric that combines them into a single number. The per-class  $F_1$  score is the harmonic mean of precision and recall:

$$F_1 = 2 \frac{Precision * Recall}{Precision + Recall}$$

As with precision, there are a number of other versions of  $F_1$  that are used in multinomial classification. The micro and weighted  $F_1$  is computed the same as with precision, but with the per-class  $F_1$  replacing the per-class precision. However, the macro  $F_1$  is computed a little differently. The precision and recall are computed by summing the TP, FN, and FP across all classes, and then using them in the standard formulas.

- **Hamming Loss:** The proportion of predictions that were incorrectly classified and is equivalent to  $1 - accuracy$ . This means a Hamming loss score of 0 is a perfect classifier. A score of  $\frac{k-1}{k}$  is a random classifier for a balanced dataset, and 1.0 is a perfectly incorrect classifier.
- **Kappa Score:** Cohen's  $\kappa$  coefficient is a statistic that measures inter-annotator agreement. This function computes Cohen's  $\kappa$ , a score that expresses the level of agreement between two annotators on a classification problem. It is defined as:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

$p_o$  is the empirical probability of agreement on the class assigned to any sample (the observed agreement ratio).  $p_e$  is the expected agreement when both annotators assign classes randomly.  $p_e$  is estimated using a per-annotator empirical prior over the class.

- **Precision (weighted, macro or micro):** This is the proportion of a class that was predicted to be in a given class and are actually in that class. In multinomial classification, it is common to report the precision for each class and this is called the per-class precision. It is computed using the same approach use in binary classification. For example,  $\frac{TP}{TP+FP}$ , but only the class under consideration is used. A value of 1 means that the classifier was able to perfectly predict, for that class. A value of 0 means that the classifier was never correct, for that class. There are three other versions of precision that are used in multinomial classification and they are weighted, macro and micro-precision. Weighted precision,  $P_w$ , combines the per-class precision by the number of true classes:

$$P_w = W_1 P_1 + \dots + W_n P_n$$

$W_i$  is the proportion of the true classes in class  $i$   $P_i$  is the per-class precision for the  $i^{th}$  class. The macro-precision,  $P_m$ , is the mean of all the per-class,  $P_i$ , precisions.

$$P_m = \frac{1}{n} \sum_i P_i$$

The micro-precision,  $P_\mu$ , is the same as the accuracy, micro-recall, and micro  $F_1$ .

- **Recall (weighted, macro or micro):** This is the proportion of the True class predictions that were correctly predicted over the number of True predictions (correct or incorrect)  $\frac{TP}{TP+FN}$ . This is also known as the True Positive Rate (TPR) or Sensitivity. In multinomial classification, it is common to report the recall for each class and this is called the micro-recall. It is computed using the same approach as in the case of binary classification, but is reported for each class. A recall of 1 is perfect recall, 0 is “bad” recall.

As with precision, there are three other versions of recall that are used in multinomial classification. They are weighted, macro and micro-recall. The definitions are the same except the per-class recall replaces the per-class precision in the preceding equations.

Generally, several of these metrics are used in combination to describe the performance of a multinomial classification model.

The prevailing charts and plots for multinomial classification are the Precision-Recall Curve, the ROC curve, the Lift Chart, the Gain Chart, and the Confusion Matrix. These are inter-related with preceding metrics, and are common across most multinomial classification literature.

For multinomial classification you can view the following using `show_in_notebook()`:

- `confusion_matrix`: A matrix of the number of actual versus predicted values for each class, see [\[Read More\]](#).
- `f1_by_label`: Harmonic mean of the precision and recall by class metrics. Compute  $F_1$  for each class, see [\[Read More\]](#)
- `jaccard_by_label`: Computes the similarity for each class distribution, see [\[Read More\]](#).
- `pr_curve`: A plot of a precision versus recall (the proportion of positive class predictions that were correct versus the proportion of positive class objects that were correctly identified), see [\[Read More\]](#).
- `precision_by_label`: It considers one class as a positive class and rest as negative. Compute precision for each, precision numbers in this example, see [\[Read More\]](#).
- `recall_by_label`: It considers one class as a positive class and rest as negative. Compute recall for each, recall numbers in this example, [\[Read More\]](#).
- `roc_curve`: A plot of a true positive rate versus a false positive rate (recall vs the proportion of negative class objects that were identified incorrectly), see [\[Read More\]](#).

To generate all of these metrics and charts for a list of multinomial classification models on the test dataset, you can run the following:

```

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from ads.common.model import ADSModel
from ads.common.data import ADSData
from ads.evaluations.evaluator import ADSEvaluator

seed = 42

X, y = make_classification(
    n_samples=10000, n_features=25, n_classes=3, flip_y=0.1, n_clusters_per_class=1
)

trainx, testx, trainy, testy = train_test_split(X, y, test_size=0.30, random_state=seed)

lr_multi_clf = LogisticRegression(
    random_state=0, solver="lbfgs", multi_class="multinomial"
).fit(trainx, trainy)

rf_multi_clf = RandomForestClassifier(n_estimators=10).fit(trainx, trainy)

multi_lr_model = ADSModel.from_estimator(lr_multi_clf)
multi_rf_model = ADSModel.from_estimator(rf_multi_clf)

evaluator = ADSEvaluator(
    ADSData(testx, testy),
    models=[multi_lr_model, multi_rf_model],
)

print(evaluator.metrics)

```

To use `ADSEvaluator`, models have to be converted into `ADSModel` types.

The `ADSModel` class in the ADS package has a `from_estimator` function that takes as input a fitted estimator and converts it into an `ADSModel` object. With classification, you have to pass the class labels in the `class` argument too. The `ADSModel` object is used for evaluation using the `ADSEvaluator` object.

To show all of the metrics in a table, run:

```
evaluator.metrics
```

```
evaluator.show_in_notebook()
```

Multinomial classification includes the following metrics:

- `accuracy`: The number of correctly classified examples divided by total examples.
- `hamming_loss`:  $1 - \text{accuracy}$

Evaluation Metrics (testing data):

	LogisticRegression	RandomForestClassifier
accuracy	0.9333	0.9333
hamming_loss	0.06667	0.06667
kappa_score_	0.8964	0.8971
precision_weighted	0.9381	0.9409
precision_micro	0.9333	0.9333
recall_weighted	0.9333	0.9333
recall_micro	0.9333	0.9333
f1_weighted	0.9338	0.9326
f1_micro	0.9333	0.9333

Fig. 5: Evaluator Metrics

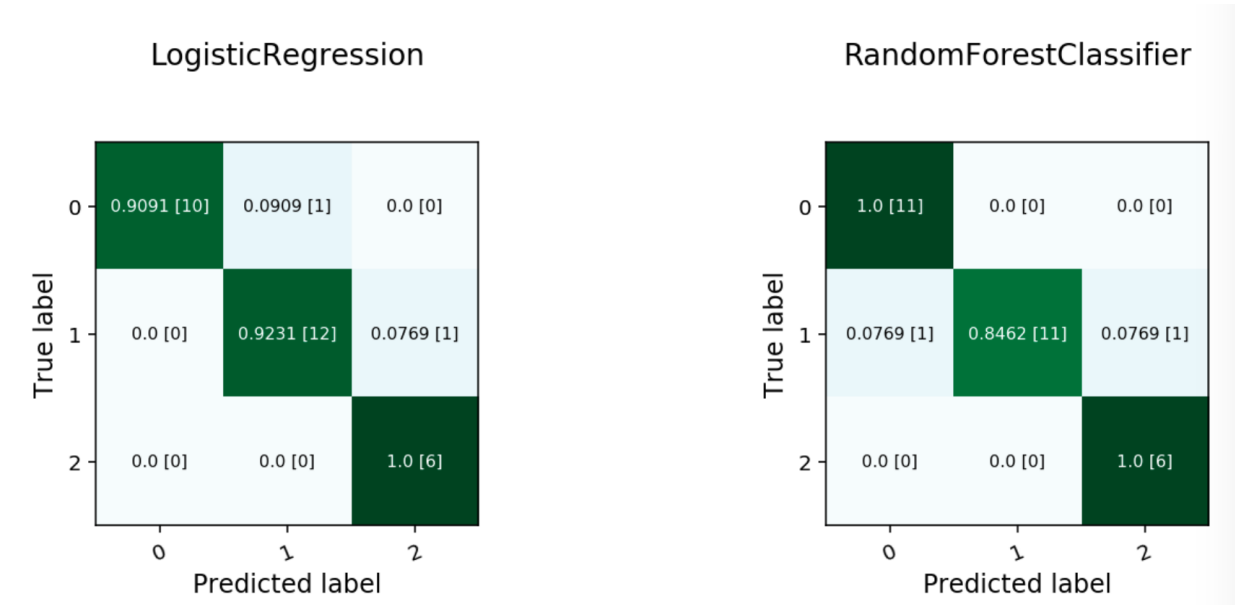


Fig. 6: Multinomial Confusion Matrix

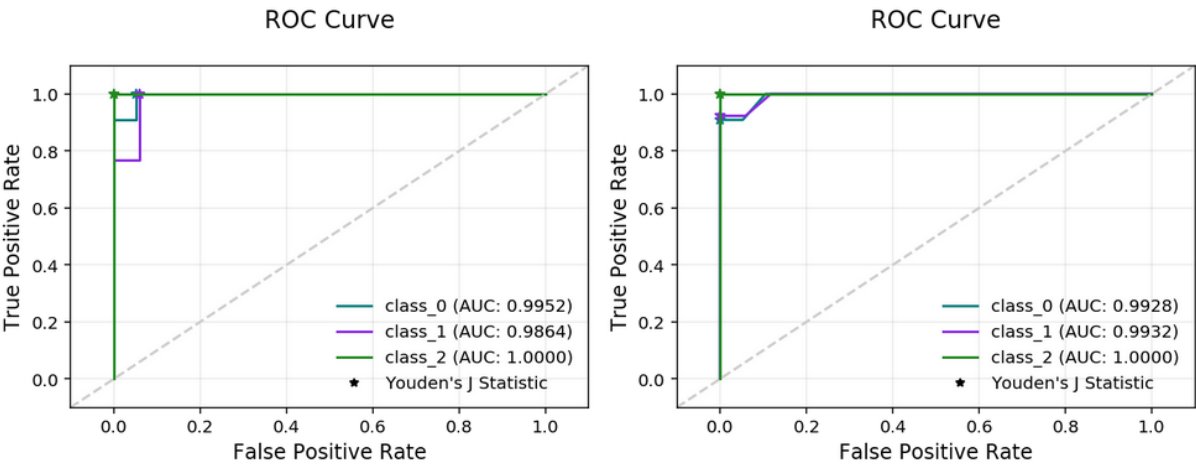


Fig. 7: Multinomial ROC Curve

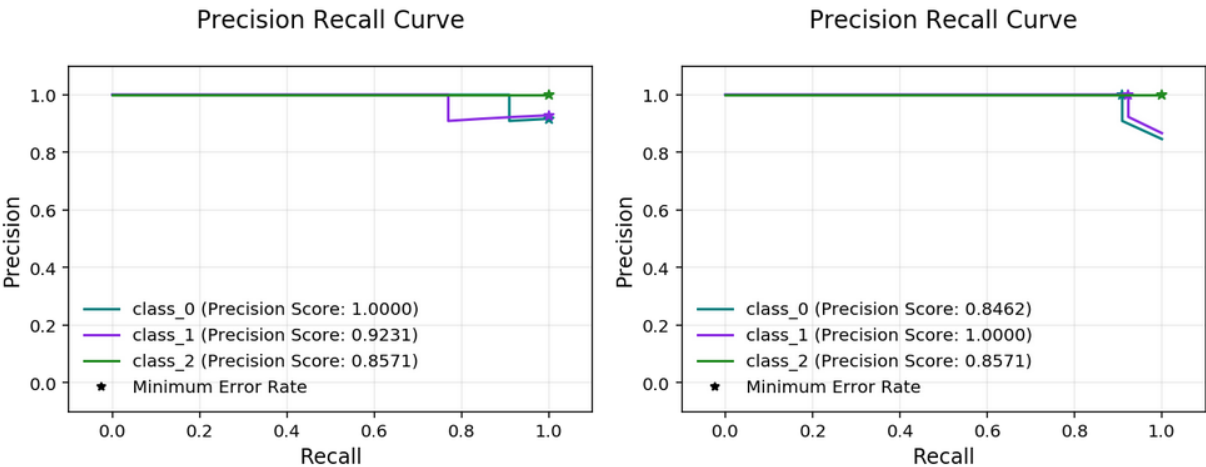


Fig. 8: Multinomial Precision Recall Curve

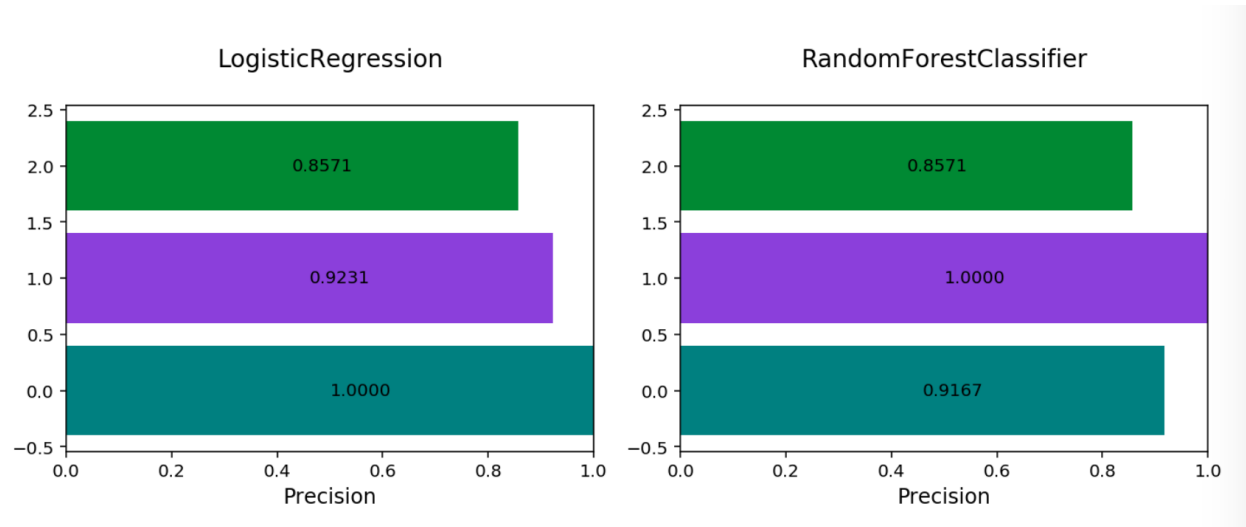


Fig. 9: Multinomial Precision By Class

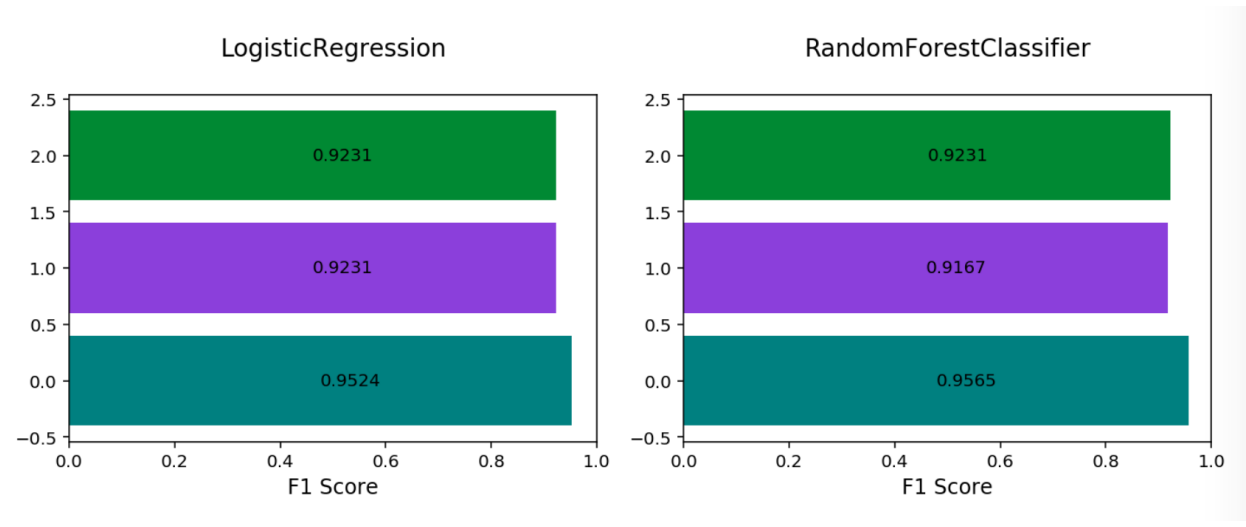


Fig. 10: Multinomial F1 By Class

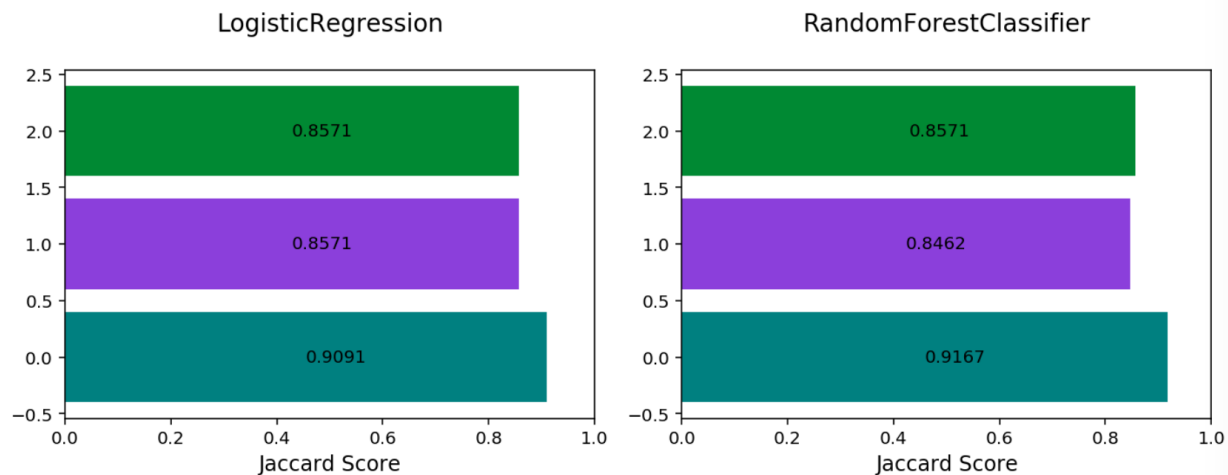


Fig. 11: Multinomial Jaccard By Class

- **precision\_weighted**: The weighted average of `precision_by_label`. Weights are proportional to the number of true instances for each class.
- **precision\_micro**: Global precision. Calculated by using global true positives and false positives.
- **recall\_weighted**: The weighted average of `recall_by_label`. Weights are proportional to the number of true instances for each class.
- **recall\_micro**: Global recall. Calculated by using global true positives and false negatives.
- **f1\_weighted**: The weighted average of `f1_by_label`. Weights are proportional to the number of true instances for each class.
- **f1\_micro**: Global  $F_1$ . It is calculated using the harmonic mean of micro precision and recall metrics.

All of these metrics can be computed directly from the confusion matrix.

If the preceding metrics don't include the specific metric you want to use, maybe an F2 score, simply add it to your evaluator object as in this example:

```
from ads.evaluations.evaluator import ADSEvaluator
evaluator = ADSEvaluator(test, models=[modelA, modelB, modelC modelD])

from sklearn.metrics import fbeta_score
def F2_Score(y_true, y_pred):
    return fbeta_score(y_true, y_pred, 2)
evaluator.add_metrics([F2_Score], ["F2 Score"])
evaluator.metrics
```

### 11.4.4 Regression

Regression is a type of modeling wherein the output is continuous. For example, price, height, sales, length. These models have their own specific metrics that help to benchmark the model. How close is close enough?

The prevailing metrics for evaluating a regression model are:

- **Explained variance score:** The variance of the model's predictions. The mean of the squared difference between the predicted values and the true mean of the data, see [\[Read More\]](#).
- **Mean absolute error (MAE):** The mean of the absolute difference between the true values and predicted values, see [\[Read More\]](#).
- **Mean squared error (MSE):** The mean of the squared difference between the true values and predicted values, see [\[Read More\]](#).
- **R-squared:** Also known as the **coefficient of determination**. It is the proportion in the data of the variance that is explained by the model, see [\[Read More\]](#).
- **Root mean squared error (RMSE):** The square root of the **mean squared error**, see [\[Read More\]](#).
- **Mean residuals:** The mean of the difference between the true values and predicted values, see [\[Read More\]](#).

The prevailing charts and plots for regression are:

- **Observed vs. predicted:** A plot of the observed, or actual values, against the predicted values output by the models.
- **Residuals QQ:** The quantile-quantile plot, shows the residuals and quantiles of a standard normal distribution. It should be close to a straight line for a good model.
- **Residuals vs observed:** A plot of residuals vs observed values. This should not carry a lot of structure in a good model.
- **Residuals vs. predicted:** A plot of residuals versus predicted values. This should not carry a lot of structure in a good model.

This code snippet demonstrates how to generate the above metrics and charts. The data has to be split into a testing and training set with the features in `X_train` and `X_test` and the responses in `y_train` and `y_test`.

```
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestClassifier

from ads.common.model import ADSModel
from ads.common.data import ADSData
from ads.evaluations.evaluator import ADSEvaluator

seed = 42

X, y = make_regression(n_samples=100000, n_features=10, n_informative=2, random_state=42)
trainx, testx, trainy, testy = train_test_split(X, y, test_size=0.3, random_state=seed)

lin_reg = LinearRegression().fit(trainx, trainy)

lasso_reg = Lasso(alpha=0.1).fit(trainx, trainy)
```

(continues on next page)



(continued from previous page)

```
lin_reg_model = ADSModel.from_estimator(lin_reg)
lasso_reg_model = ADSModel.from_estimator(lasso_reg)

reg_evaluator = ADSEvaluator(
    ADSData(testx, testy), models=[lin_reg_model, lasso_reg_model]
)

print(reg_evaluator.metrics)
```

To show all of the metrics in a table, run:

```
evaluator.metrics
```

#### Evaluation Metrics (testing data):

	LinearRegression	Lasso
<b>r2_score</b>	0.5982	0.5749
<b>mse</b>	24.94	26.39
<b>explained_variance</b>	0.5997	0.5758
<b>mae</b>	3.326	3.348

Fig. 12: Evaluator Metrics

To show all of the charts, run:

```
evaluator.show_in_notebook()
```

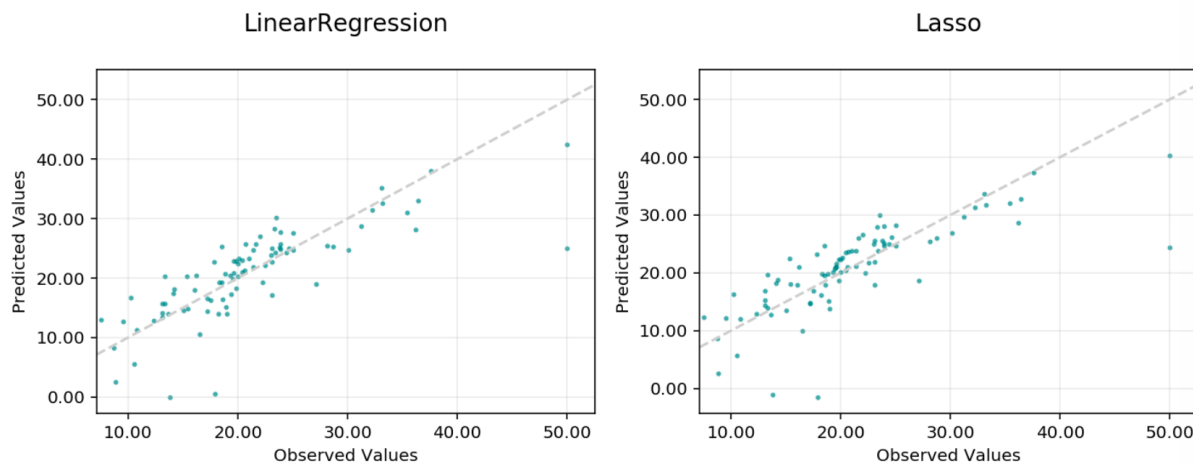


Fig. 13: Observed vs Predicted

This code snippet demonstrates how to add a custom metric, *Number Correct*, to the evaluator.

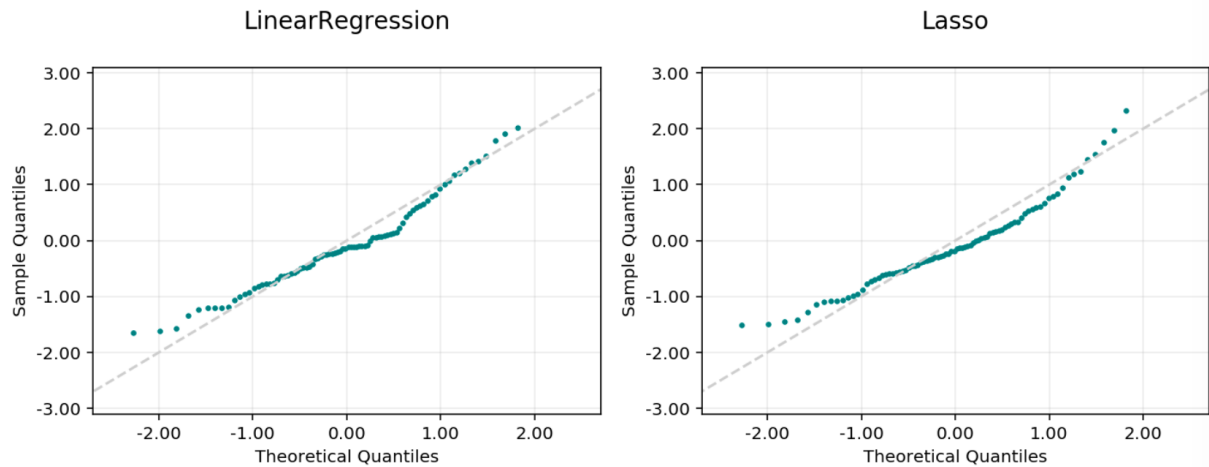


Fig. 14: Residual Q-Q Plot

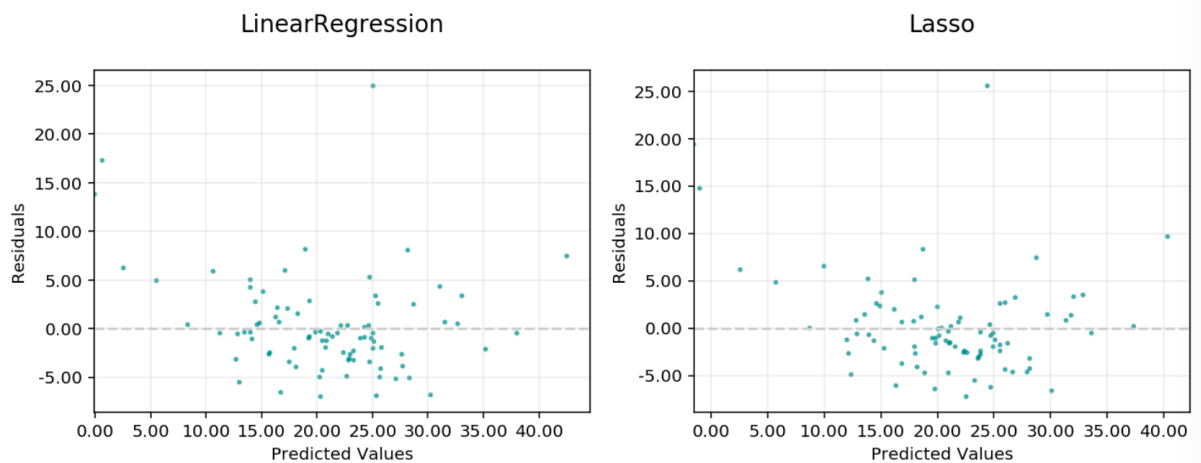


Fig. 15: Residual vs Predicted

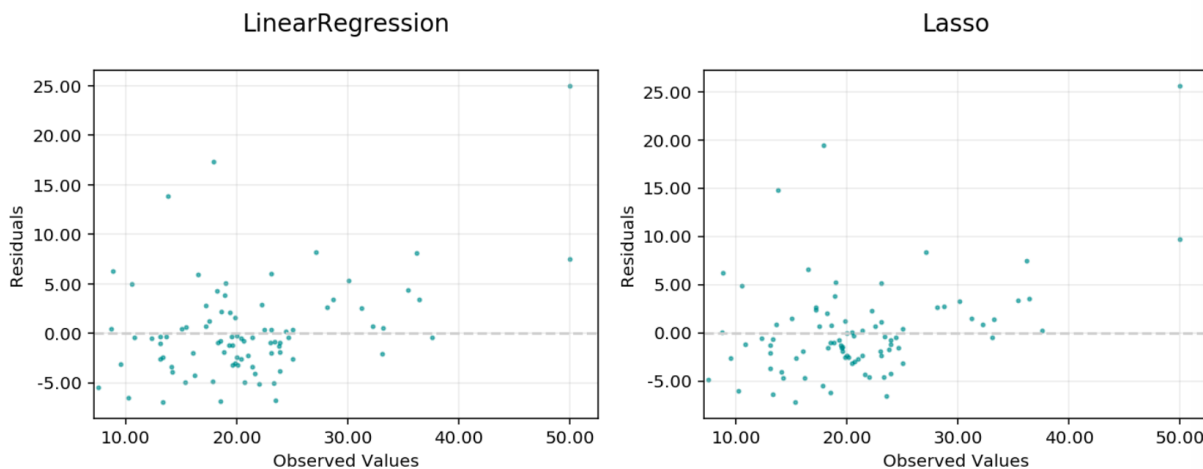


Fig. 16: Residual vs Observed

```
from ads.evaluations.evaluator import ADSEvaluator
evaluator = ADSEvaluator(test, models=[modelA, modelB, modelC, modelD])

def num_correct(y_true, y_pred):
    return sum(y_true == y_pred)
evaluator.add_metrics([num_correct], ["Number Correct"])
evaluator.metrics
```

## 11.5 Model Explainability

### Prerequisites

- Currently Oracle AutoML and MLX libraries are available only via Data Science Conda Packs.
- See [here](#) for supported conda packs
- To install conda packs locally, see [Working with Conda Packs](#)

Machine learning and deep learning are becoming ubiquitous due to:

- The ability to solve complex problems in a variety of different domains.
- The growth in the performance and efficiency of modern computing resources.
- The widespread availability of large amounts of data.

However, as the size and complexity of problems continue to increase, so does the complexity of the machine learning algorithms applied to these problems. The inherent and growing complexity of machine learning algorithms limits the ability to understand what the model has learned or why a given prediction was made, acting as a barrier to the adoption of machine learning. Additionally, there may be legal or regulatory requirements to be able to explain the outcome of a prediction from a machine learning model, resulting in the use of biased models at the cost of accuracy.

Machine learning explainability (MLX) is the process of explaining and interpreting machine learning and deep learning models.

MLX can help machine learning developers to:

- Better understand and interpret the model's behavior.
  - Which features does the model consider important?
  - What is the relationship between the feature values and the target predictions?
- Debug and improve the quality of the model.
  - Did the model learn something unexpected?
  - Does the model generalize or did it learn something specific to the training dataset?
- Increase trust in the model and confidence in deploying the model.

MLX can help users of machine learning algorithms to:

- Understand why the model made a certain prediction.
  - Why was my bank loan denied?

Some useful terms for MLX:

- **Explainability:** The ability to explain the reasons behind a machine learning model's prediction.
- **Global Explanations:** Understand the general behavior of a machine learning model as a whole.
- **Interpretability:** The level at which a human can understand the explanation.
- **Local Explanations:** Understand why the machine learning model made a specific prediction.
- **Model-Agnostic Explanations:** Explanations treat the machine learning model and feature pre-processing as a black box, instead of using properties from the model to guide the explanation.
- **WhatIf Explanations:** Understand how changes in the value of features affects the model's prediction.

The ADS explanation module provides interpretable, model-agnostic, local and global explanations.

## 11.5.1 Accumulated Local Effects

### 11.5.1.1 Overview

Similar to Partial Dependence Plots (PDP), Accumulated Local Effects (ALE) is a model-agnostic global explanation method that evaluates the relationship between feature values and target variables. However, in the event that features are highly correlated, PDP may include unlikely combinations of feature values in the average prediction calculation due to the independent manipulation of feature values across the marginal distribution. This lowers the trust in the PDP explanation when features have strong correlation. Unlike PDP, ALE handles feature correlations by averaging and accumulating the difference in predictions across the conditional distribution, which isolates the effects of the specific feature. This comes at the cost of requiring a larger number of observations and a near uniform distribution of those observations so that the conditional distribution can be reliably determined.

### 11.5.1.2 Description

ALE highlights the effects that specific features have on the predictions of a machine learning model by partially isolating the effects of other features. Therefore, it tends to be robust against correlated features. The resulting ALE explanation is centered around the mean effect of the feature, such that the main feature effect is compared relative to the average prediction of the data.

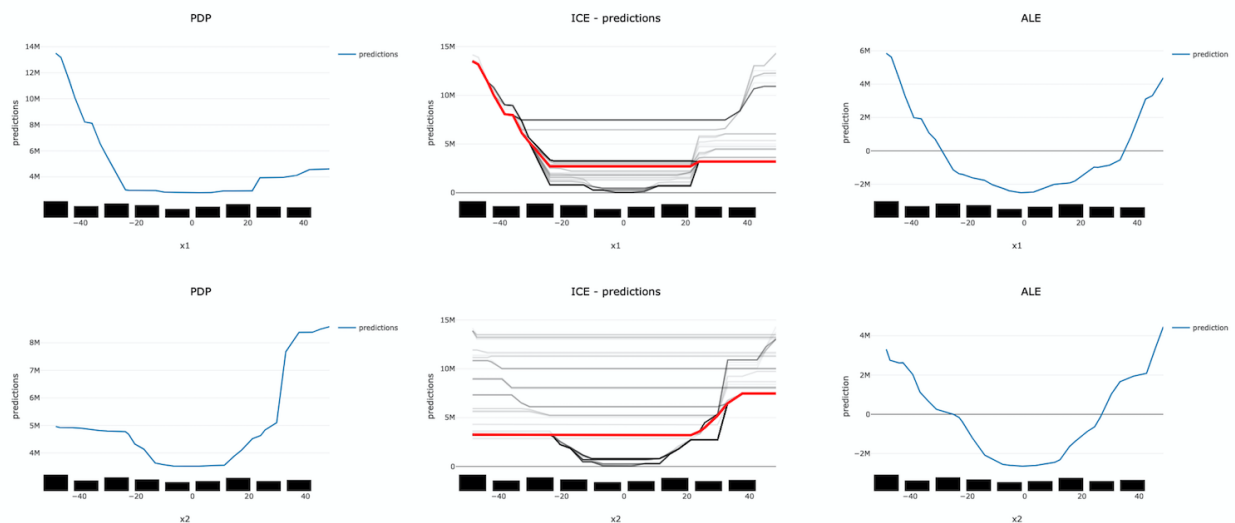
Correlated features can negatively affect the quality of many explanation techniques. Specifically, many challenges arise when the black-box model is used to make predictions on unlikely artificial data. That is data that fall outside of the expected data distribution but are used in an explanation because they are not independent and the technique is not sensitive to this possibility. This can occur, for example, when the augmented data samples are not generated

according to the feature correlations or the effects of other correlated features are included in the evaluation of the feature of interest. Consequently, the resulting explanations may be misleading. In the context of PDP, the effect of a given feature may be heavily biased by the interactions with other features.

To address the issues associated with correlated features, ALE:

- Uses the conditional distribution of the feature of interest to generate augmented data. This tends to create more realistic data than using marginal distribution. This helps to ensure that evaluated feature values, e.g.,  $x_i$ , are only compared with instances from the dataset that have similar values to  $x_i$ .
- Calculates the average of the differences in model predictions over the augmented data, instead of the average of the predictions themselves. This helps to isolate the effect of the feature of interest. For example, assuming we are evaluating the effect of a feature at value  $x_i$ , ALE computes the average of the difference in model predictions of the values in the neighborhood of  $x_i$ . That is, that observation within  $x_i \pm$  that meet the conditional requirement. This helps to reduce the effects of correlated features.

The following example demonstrates the challenges with accurately evaluating the effect of a feature on a model's predictions when features are highly correlated. Let us assume that features  $x_1$  and  $x_2$  are highly correlated. We can artificially construct  $x_2$  by starting with  $x_1$  and adding a small amount of random noise. Further assume that the target value is the product of these two features (e.g.,  $y = x_1 * x_2$ ). Since  $x_1$  and  $x_2$  are almost identical, the target value has a quadratic relationship with them. A decision tree is trained on this dataset. Then different explanation techniques, PDP (first column), ICE (second column), and ALE (third column), are used to evaluate the effect of the features on the model predictions. Features  $x_1$  and  $x_2$  are evaluated in the first and second row, respectively. The following image demonstrates that PDP is unable to accurately identify the expected relationship due to the assumption that the features are not correlated. An examination of the ICE plots reveals the quadratic relationship between the features and the target. However, when taken as an aggregate, this effect disappears. In contrast, ALE is able to properly capture the isolated effect of each feature, highlighting the quadratic relationship.



The following summarizes the steps in computing ALE explanation (note: MLX supports one-feature ALE):

- Start with a trained model.
- Select a feature to explain (for example, one of the important features identified in the global feature importance explanations).
- Compute the intervals of the selected feature to define the upper and lower bounds used to compute the difference in model predictions when the feature is increased or decreased.
  - Numerical features: using the selected feature's value distribution extracted from the train dataset, MLX selects multiple different intervals from the feature's distribution to evaluate (e.g., based on percentiles).

The number of intervals to use and the range of the feature's distribution to consider are configurable.

- Categorical features: since ALE computes the difference in model predictions between an increase and decrease in a feature's value, features must have some notion of order. This can be challenging for categorical features, as there may not be a notion of order (e.g., eye color). To address this, MLX estimates the order of categorical feature values based on a categorical feature encoding technique. MLX provides multiple different encoding techniques based on the input data (e.g., `distance_similarity`: computes a similarity matrix between all categorical feature values and the other feature values, and orders based on similarity. Target-based approaches estimate the similarity/order based on the relationship of categorical feature values with the target variable. The supported techniques include, `target`, James-Stein encoding, `jamesstein`, Generalized Linear Mixed Model encoding, `glmm`, M-estimate encoding, `mestimate`, and Weight of Evidence encoding, `woe`. The categorical feature value order is then used to compute the upper (larger categorical value) and lower (smaller categorical value) bounds for the selected categorical feature.
- For each interval, MLX approximates the conditional distribution by identifying the samples that are in the neighborhood of the sample of interest. It then calculates the difference in the model prediction when the selected feature's value of the samples is replaced by the upper and lower limits of the interval. If  $N$  different intervals are selected from the feature's distribution, this process results in  $2N$  different augmented datasets. It is  $2N$  as each selected feature of the sample are replaced with the upper and lower limits of the interval. The model inference then generates  $2N$  different model predictions, which are used to calculate the  $N$  differences.
- The prediction differences within each interval are averaged and accumulated in order, such that the ALE of a feature value that lies in the  $k$ -th interval is the sum of the effects of the first through the  $k$ -th interval.
- Finally, the accumulated feature effects at each interval is centered, such that the mean effect is zero.

### 11.5.1.3 Interpretation

- Continuous or discrete numerical features: Visualized as line graphs. Each line represents the change in the model prediction when the selected feature has the given value compared to the average prediction. For example, an ALE value of  $\pm b$  at  $x_j = k$  indicates that when the value of feature  $j$  is equal to  $k$ , the model prediction is higher/lower by  $b$  compared to the average prediction. The x-axis shows the selected feature values and the y-axis shows the delta in the target prediction variable relative to the average prediction (e.g., the prediction probability for classification tasks and the raw predicted values for regression tasks).
- Categorical features: Visualized as vertical bar charts. Each bar represents the change in the model prediction when the selected feature has the given value compared to the average prediction. The interpretation of the value of the bar is similar to continuous features. The x-axis shows the different categorical values for the selected feature and the y-axis shows the change in the predicted value relative to the average prediction. This would be the prediction probability for classification tasks and the raw predicted values for regression tasks.

### 11.5.1.4 Limitations

There is an increased computational cost for performing an ALE analysis because of the large number of models that need to be computed relative to PDP. On a small dataset, this is generally not an issue. However, on larger datasets it can be. It is possible to parallelize the process and to also compute it in a distributed manner.

The main disadvantage comes from the problem of sparsity of data. There needs to be sufficient number of observations in each neighborhood that is used in order to make a reasonable estimation. Even with large dataset this can be problematic if the data is not uniformly sampled, which is rarely the case. Also, with higher dimensionality the problem is made increasingly more difficult because of this curse of dimensionality.

Depending on the class of model that is being used, it is common practice to remove highly correlated features. In these cases there is some rationale to using a PDP for interpretation. However, if there is correlation in the data and the sampling of the data is suitable for an ALE analysis, it may be the preferred approach.

### 11.5.1.5 Examples

The following is a purposefully extreme, but realistic, example that demonstrates the effects of highly correlated features on PDP and ALE explanations. The data set has three columns, `x1`, `x2` and `y`.

- `x1` is generated from a uniform distribution with a range of `[-5, 5]`.
- `x2` is `x1` with some noise. `x1` and `x2` are highly correlated for illustration purposes.
- `y` is our target which is generated from an interaction term of `x1 * x2` and `x2`.

This model is trained using a Sklearn RegressorMixin model and wrapped in an `ADSModel` object. Please note that the ADS model explainers work with any model that is wrapped in an `ADSModel` object.

```
import numpy as np
import pandas as pd
from ads.dataset.factory import DatasetFactory
from ads.common.model import ADSModel
from sklearn.base import RegressorMixin

x1 = (np.random.rand(500) - 0.5) * 10
x2 = x1 + np.random.normal(loc=0, scale=0.5, size=500)
y = x1 * x2

correlated_df = pd.DataFrame(np.stack((x1, x2, y), axis=1), columns=['x1', 'x2', 'y'])
correlated_ds = DatasetFactory.open(correlated_df, target='y')

correlated_train, _ = correlated_ds.train_test_split(test_size=0)

class CorrelatedRegressor(RegressorMixin):
    """
    implement the true model
    """
    def fit(self, X=None, y=None):
        self.y_bar_ = X.iloc[:, 0].to_numpy() * X.iloc[:, 1].to_numpy() + X.iloc[:, 1].
        ↪to_numpy()

    def predict(self, X=None):
        return X.iloc[:, 0].to_numpy() * X.iloc[:, 1].to_numpy() + X.iloc[:, 1].to_
        ↪numpy()

# train a RegressorMixin model
# Note that the ADSExplainer below works with any model (classifier or
# regressor) that is wrapped in an ADSModel
correlated_regressor = CorrelatedRegressor()
correlated_regressor.fit(correlated_train.X, correlated_train.y)

# Build ads models from ExtraTrees regressor
correlated_model = ADSModel.from_estimator(correlated_regressor, name="TrueModel")

# Create the ADS explainer object, which is used to construct
# global and local explanation objects. The ADSExplainer takes
```

(continues on next page)

(continued from previous page)

```

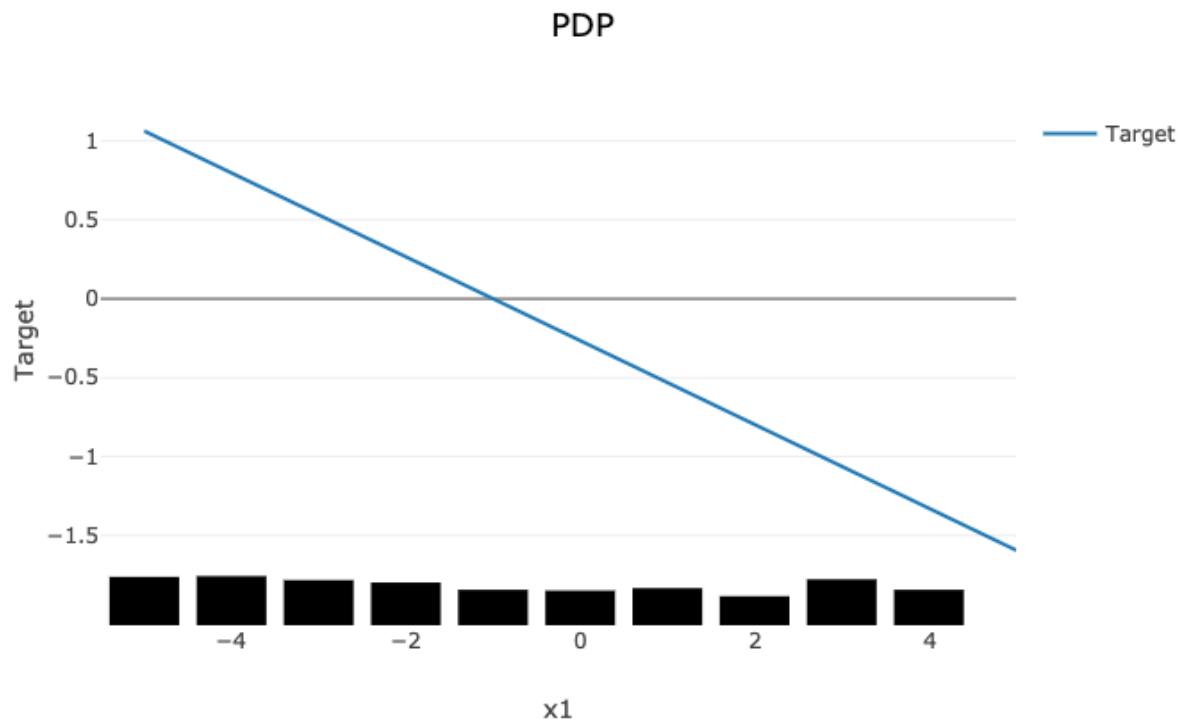
# as input the model to explain and the train/test dataset
from ads.explanations.explainer import ADSExplainer
correlated_explainer = ADSExplainer(correlated_train, correlated_model, training_
    ↪ data=correlated_train)

# With ADSExplainer, create a global explanation object using
# the MLXGlobalExplainer provider
from ads.explanations.mlx_global_explainer import MLXGlobalExplainer
correlated_global_explainer = correlated_explainer.global_
    ↪ explanation(provider=MLXGlobalExplainer())

# A summary of the global accumulated local effects explanation
# algorithm and how to interpret the output
correlated_global_explainer.accumulated_local_effects_summary()

# compute a PDP between x1 and the target, y
pdp_x1 = correlated_global_explainer.compute_partial_dependence("x1")
pdp_x1.show_in_notebook()

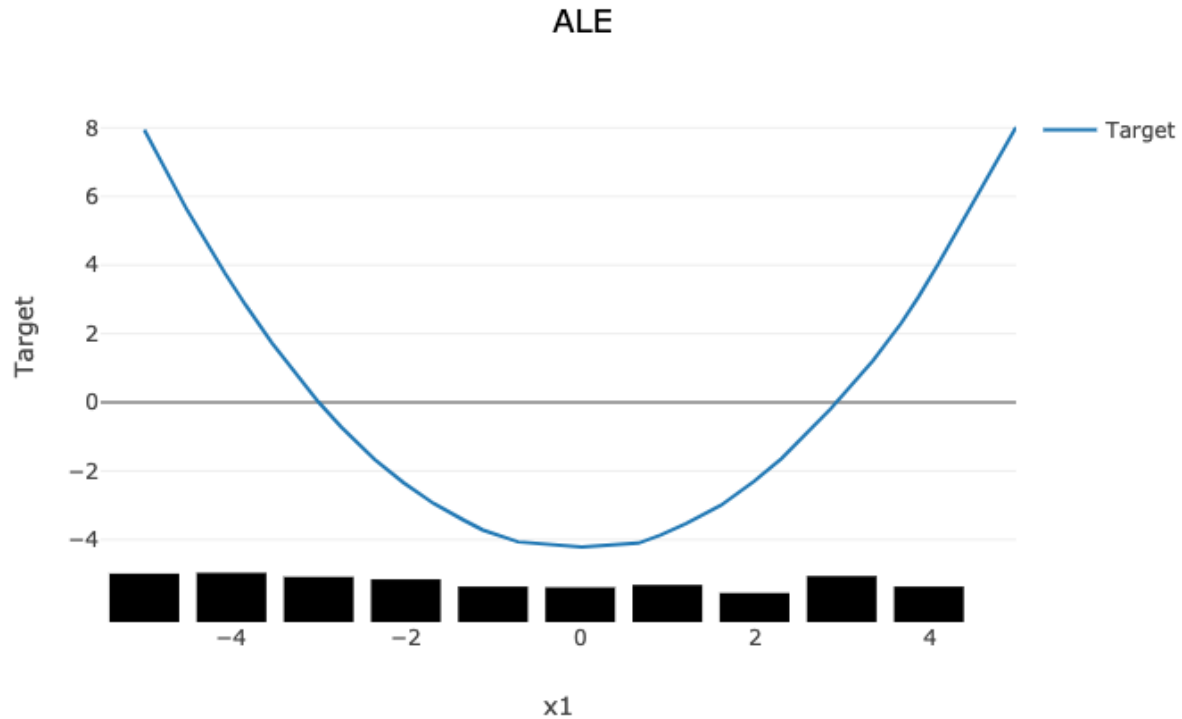
```



The PDP plot shows a rug plot of the actual  $x_1$  values along the x-axis and the relationship between  $x_1$  and  $y$  appears as a line. However, it is known that the true relationship is not linear.  $y$  is the product of  $x_1$  and  $x_2$ . Since  $x_2$  is nearly identical to  $x_1$ , effectively the relationship between  $x_1$  and  $y$  is quadratic. The high level of correlation between  $x_1$  and  $x_2$  violates one of the assumptions of the PDP. As demonstrated, the bias created by this correlation results in a poor representation of the global relationship between  $x_1$  and  $y$ .



```
# Compute the ALE on x1
ale_x1 = correlated_global_explainer.compute_accumulated_local_effects("x1")
ale_x1.show_in_notebook()
```



In comparison, the ALE plot does not have as strong a requirement that the features are uncorrelated. As such, there is very little bias introduced when they are. The following ALE plot demonstrates that it is able to accurately represent the relationship between  $x_1$  and  $y$  as being quadratic. This is due to the fact that ALE uses the conditional distribution of these two features. This can be thought of as only using those instances where the values of  $x_1$  and  $x_2$  are close.

In general, ALE plots are unbiased with correlated features as they use conditional probabilities. The PDP method uses the marginal probability and that can introduce a bias when there are highly correlated features. The advantage is that when the data is not rich enough to adequately determine all of the conditional probabilities or when the features are not highly correlated, it can be an effective method to assess the global impact of a feature in a model.

### 11.5.1.6 References

- [Accumulated Local Effects \(ALE\) Plot](#)
- [Visualizing the effects of predictor variables in black box supervised learning models](#)

## 11.5.2 Feature Dependence Explanations

### 11.5.2.1 Overview

Feature Dependence Explanations (PDP and ICE) are model-agnostic global explanation methods that evaluate the relationship between feature values and model target predictions.

### 11.5.2.2 Description

PDP and ICE highlight the marginal effect that specific features have on the predictions of a machine learning model. These explanation methods visualize the effects that different feature values have on the model's predictions.

These are the main steps in computing PDP or ICE explanations:

- Start with a trained machine learning model.
- Select a feature to explain (for example, one of the important features identified in the global feature permutation importance explanations.)
- Using the selected feature's value distribution extracted from the training dataset, ADS selects multiple different values from the feature's distribution to evaluate. The number of values to use and the range of the feature's distribution to consider are configurable.
- ADS replaces every sample in the provided dataset with the same feature value from the feature distribution and computes the model inference on the augmented dataset. This process is repeated for all of the selected values from the feature's distribution. If  $N$  different values are selected from the feature's distribution, this process results in  $N$  different datasets. Each with the selected feature having the same value for all samples in the corresponding dataset. The model inference then generates  $N$  different model predictions, each with  $M$  values (one for each sample in the augmented dataset.)
- For ICE, the model predictions for each augmented sample in the provided dataset are considered separately when the selected feature's value is replaced with a value from the feature distribution. This results in  $N \times M$  different values.
- For PDP, the average model prediction is computed across all augmented dataset samples. This results in  $N$  different values (each an average of  $M$  predictions).

The preceding is an example of one-feature PDP and ICE explanations. PDP also supports two-feature explanations while ICE only supports one feature. The main steps of the algorithm are the same though the explanation is computed on two features instead of one.

- Select two features to explain.
- ADS computes the cross-product of values selected from the feature distributions to generate a list of different value combinations for the two selected features. For example, assuming we have selected  $N$  values from the feature distribution for each feature:

$$[(X_1^1, X_2^1), (X_1^1, X_2^2), \dots, (X_1^1, X_2^{N-1}), (X_1^1, X_2^N), (X_1^2, X_2^1), (X_1^2, X_2^2), \dots, (X_1^N, X_2^{N-1}), (X_1^N, X_2^N)]$$

- For each feature value combination, ADS replaces every sample in the provided set with these two feature values and computes the model inference on the augmented dataset. There are  $M$  different samples in the provided dataset and  $N$  different values for each selected feature. This results in  $N^2$  predictions from the model, each an average of  $M$  predictions.

### 11.5.2.3 Interpretation

#### 11.5.2.3.1 PDP

- One-feature
  - Continuous or discrete numerical features: Visualized as line graphs, each line represents the average prediction from the model (across all samples in the provided dataset) when the selected feature is replaced with the given value. The x-axis shows the selected feature values and the y-axis shows the predicted target (e.g., the prediction probability for classification tasks and the raw predicted values for regression tasks).
  - Categorical features: Visualized as vertical bar charts. Each bar represents the average prediction from the model (across all samples in the provided dataset) when the selected feature is replaced with the given value. The x-axis shows the different values for the selected feature and the y-axis shows the predicted target (e.g., the prediction probability for classification tasks and the raw predicted values for regression tasks).
- Two-feature
  - Visualized as a heat map. The x and y-axis both show the selected feature values. The heat map color represents the average prediction from the model (across all samples in the provided dataset) when the selected features are replaced with the corresponding values.

#### 11.5.2.3.2 ICE

- Continuous or discrete numerical features: Visualized as line graphs. While PDP shows the average prediction across all samples in the provided dataset, ICE plots every sample from the provided dataset (when the selected feature is replaced with the given value) separately. The x-axis shows the selected feature values and the y-axis shows the predicted target (for example, the prediction probability for classification tasks and the raw predicted values for regression tasks). The median value can be plotted to highlight the trend. The ICE plots can also be centered around the first prediction from the feature distribution (for example, each prediction subtracts the predicted value from the first sample).
- Categorical features: Visualized as violin plots. The x-axis shows the different values for the selected feature and the y-axis shows the predicted target (for example, the prediction probability for classification tasks and the raw predicted values for regression tasks).

Both PDP and ICE visualizations display the feature value distribution from the training dataset on the corresponding axis. For example, the one-feature line graphs, bar charts, and violin plots show the feature value distribution on the x-axis. The heat map shows the feature value distributions on the respective x-axis or y-axis.

#### 11.5.2.4 Examples

The following example generates and visualizes global partial dependence plot (PDP) and Individual Conditional Expectation (ICE) explanations on the [Titanic dataset](#). The model is constructed using the `ADS OracleAutoMLProvider` (selected model: `XGBClassifier`), however, the ADS model explainers work with any model (classifier or regressor) that is wrapped in an `ADSModel` object.

```
from ads.dataset.factory import DatasetFactory
from os import path
import requests

# Prepare and load the dataset
titanic_data_file = '/tmp/titanic.csv'
```

(continues on next page)

(continued from previous page)

```

if not path.exists(titanic_data_file):
    # fetch and save some data
    print('fetching data from web...', end=" ")
    # Data source: https://www.openml.org/d/40945
    r = requests.get('https://www.openml.org/data/get_csv/16826755/phpMYEkMl')
    with open(titanic_data_file, 'wb') as fd:
        fd.write(r.content)
    print("Done")
ds = DatasetFactory.open(
    titanic_data_file, target="survived").set_positive_class(True)
ds = ds.drop_columns(['name', 'ticket', 'cabin', 'boat',
                      'body', 'home.dest'])
ds = ds[ds['age'] != '?'].astype({'age': 'float64'})
ds = ds[ds['fare'] != '?'].astype({'fare': 'float64'})
train, test = ds.train_test_split(test_size=0.2)

# Build the model using AutoML. 'model' is a subclass of type ADSModel.
# Note that the ADSExplainer below works with any model (classifier or
# regressor) that is wrapped in an ADSModel
import logging
from ads.automl.provider import OracleAutoMLProvider
from ads.automl.driver import AutoML
ml_engine = OracleAutoMLProvider(n_jobs=-1, loglevel=logging.ERROR)
oracle_automl = AutoML(train, provider=ml_engine)
model, baseline = oracle_automl.train()

# Create the ADS explainer object, which is used to construct
# global and local explanation objects. The ADSExplainer takes
# as input the model to explain and the train/test dataset
from ads.explanations.explainer import ADSExplainer
explainer = ADSExplainer(test, model, training_data=train)

# With ADSExplainer, create a global explanation object using
# the MLXGlobalExplainer provider
from ads.explanations.mlx_global_explainer import MLXGlobalExplainer
global_explainer = explainer.global_explanation(
    provider=MLXGlobalExplainer())

# A summary of the global partial feature dependence explanation
# algorithm and how to interpret the output can be displayed with
global_explainer.partial_dependence_summary()

# Compute the 1-feature PDP on the categorical feature, "sex",
# and numerical feature, "age"
pdp_sex = global_explainer.compute_partial_dependence("sex")
pdp_age = global_explainer.compute_partial_dependence(
    "age", partial_range=(0, 1))

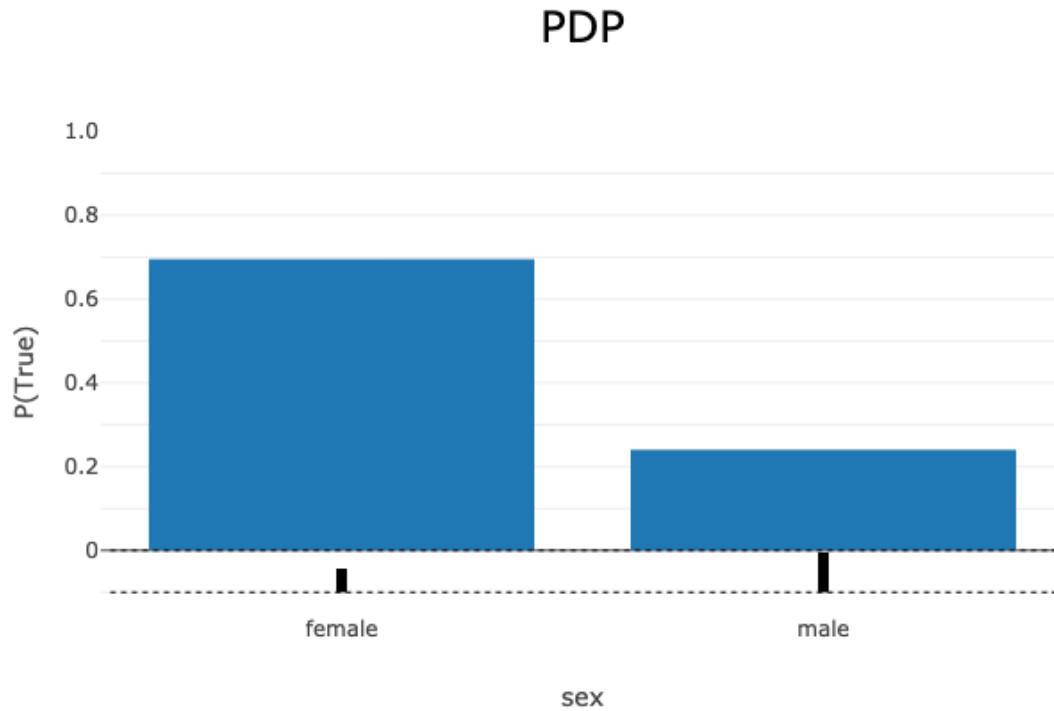
# ADS supports PDP visualizations for both 1-feature and 2-feature
# Feature Dependence explanations, and ICE visualizations for 1-feature
# Feature Dependence explanations (see "Interpretation" above)

```

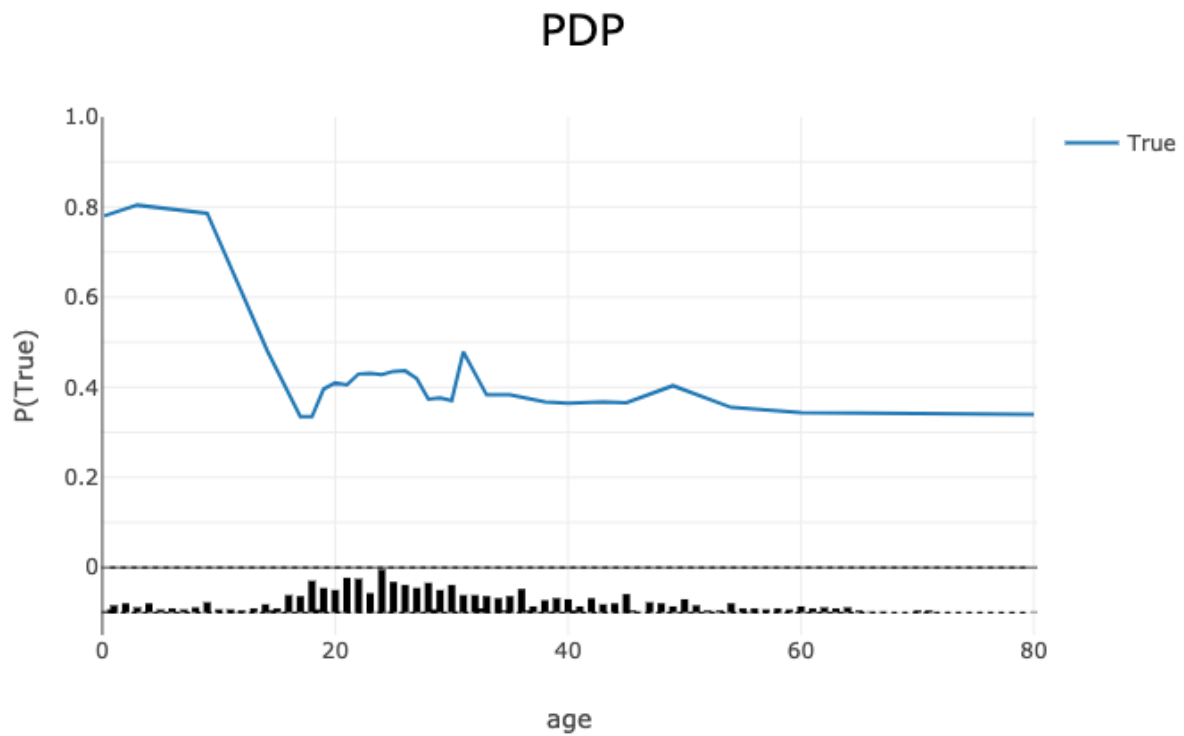
(continues on next page)

(continued from previous page)

```
# Visualize the categorical feature PDP for the True (Survived) label  
pdp_sex.show_in_notebook(labels=True)
```

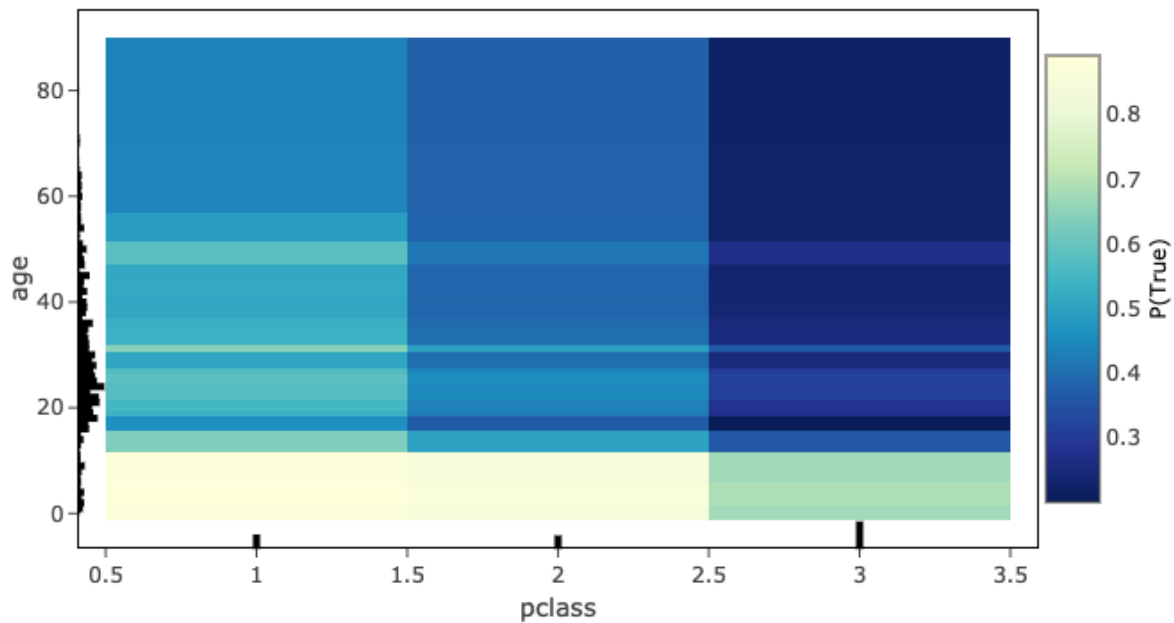


```
# Visualize the numerical feature PDP for the True (Survived) label  
pdp_age.show_in_notebook(labels=True)
```



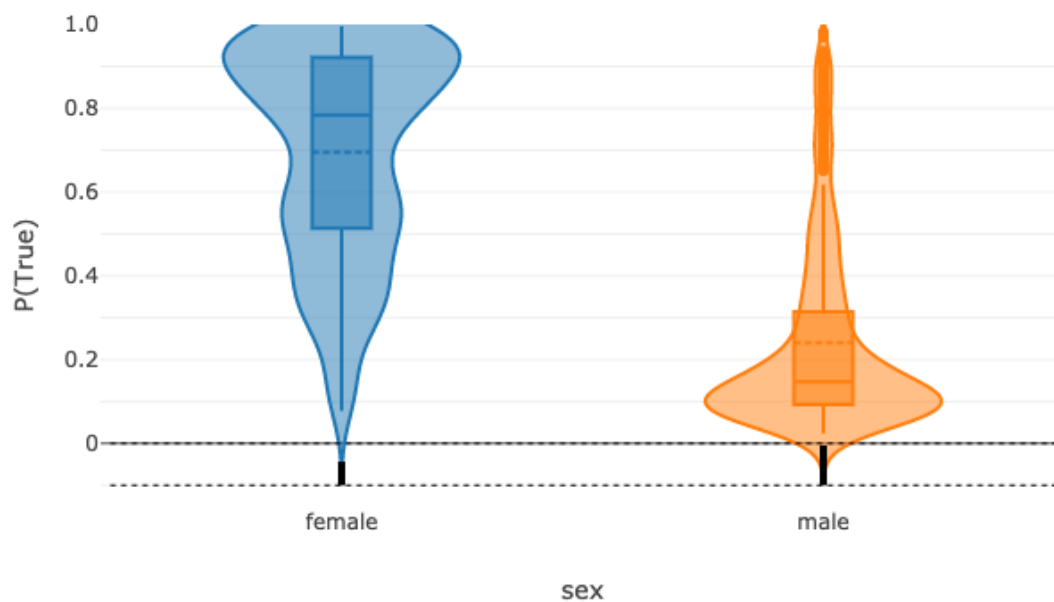
```
# Compute the 2-feature PDP on the categorical feature, "pclass", and
# numerical feature, "age"
pdp_pclass_age = global_explainer.compute_partial_dependence(
    ['pclass', 'age'], partial_range=(0, 1))
pdp_pclass_age.show_in_notebook(labels=True)
```

## PDP - True

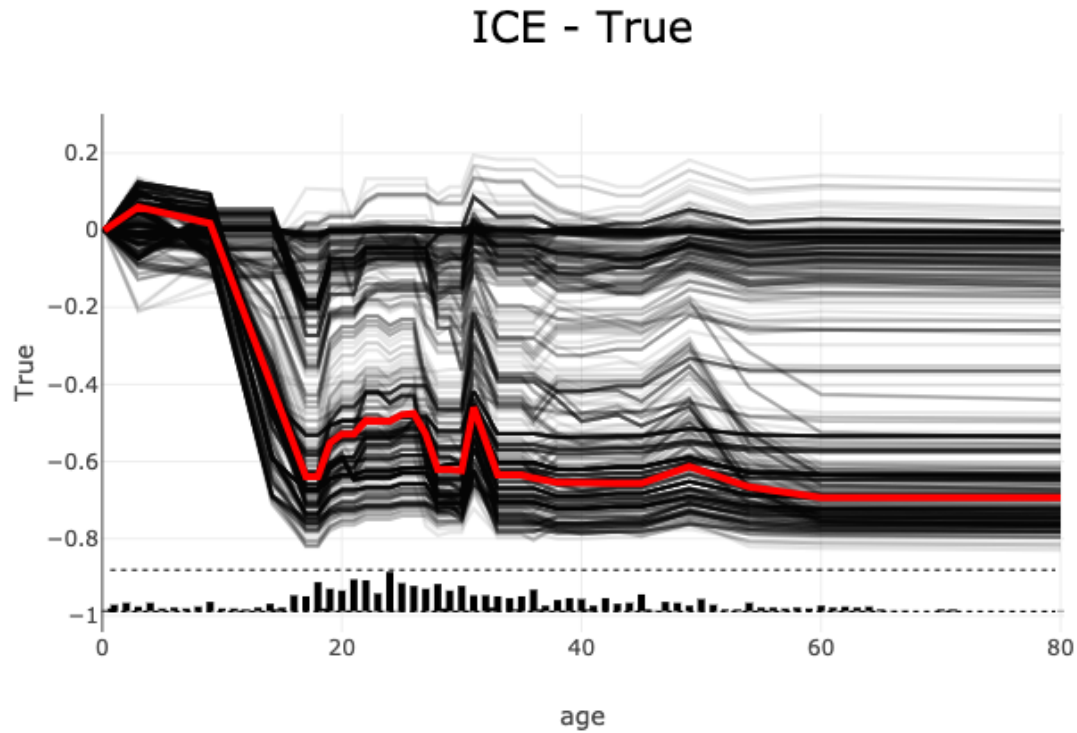


```
# Visualize the ICE plot for the categorical feature, "sex"  
pdp_sex.show_in_notebook(mode='ice', labels=True)
```

## ICE - True



```
# Visualize the ICE plot for the numerical feature, "age", and center  
# around the first prediction (smallest age)  
pdp_age.show_in_notebook(mode='ice', labels=True, centered=True)
```



```
# The raw explanation data used to generate the visualizations, as well  
# as the runtime performance information can be extracted with  
pdp_age.get_diagnostics()
```



```

{'feature_correlations': {},
 'explanation_stats': {'Runtime analysis': {'samples': {'value': [0.1648237705230713,
 1.7607676982879639],
 'work': [2, 30]},
 'samples average': 0.9627957344055176,
 'samples total': 1.9255914688110352,
 'work average': 16.0,
 'work total': 32,
 'samples throughput': 16.618270551311976,
 'samples latency': 0.06017473340034485}},
 'pdp': [{'age': 0.1667,
 'mean': [0.21962339, 0.7803766],
 'std': [0.18659413, 0.18659413]},
 {'age': 3.0,
 'mean': [0.19585957, 0.8041404],
 'std': [0.20155172, 0.20155172]},
 {'age': 9.0,
 'mean': [0.21387196, 0.786128],
 'std': [0.19405662, 0.19405662]},
 {'age': 14.172413793103445,
 'mean': [0.5195417, 0.4804583],
 'std': [0.31325987, 0.31325987]},
 {'age': 17.0, 'mean': [0.6653859, 0.3346141], 'std': [0.3234773, 0.3234773]},
 {'age': 18.0, 'mean': [0.6653859, 0.3346141], 'std': [0.3234773, 0.3234773]},
 {'age': 19.0, 'mean': [0.6038667, 0.3961334], 'std': [0.3292721, 0.3292721]},
 {'age': 20.0,
 'mean': [0.5904662, 0.40953377],
 'std': [0.32400262, 0.32400262]},
 {'age': 21.0,
 'mean': [0.59451133, 0.4054887],
 'std': [0.32557142, 0.32557142]},
 {'age': 22.0, 'mean': [0.5704925, 0.4295075], 'std': [0.3247535, 0.3247535]},
 {'age': 23.0,
 'mean': [0.56890285, 0.43109715],
 'std': [0.32589453, 0.32589453]},
 {'age': 24.0,
 'mean': [0.5722691, 0.4277309],
 'std': [0.32457417, 0.32457417]},
 {'age': 25.0,
 'mean': [0.5646265, 0.43537349],
 'std': [0.32141125, 0.32141125]},
 {'age': 26.0,
 'mean': [0.56357104, 0.43642896],
 'std': [0.32082796, 0.32082796]},
 {'age': 27.0,
 'mean': [0.58106536, 0.41893464],
 'std': [0.31745076, 0.31745076]},
 {'age': 28.0,
 'mean': [0.62635016, 0.37364992],
 'std': [0.32987198, 0.32987198]},
 {'age': 29.0,
 'mean': [0.6237644, 0.37623563],
 'std': [0.3303695, 0.3303695]},
 {'age': 30.0,
 'mean': [0.62962914, 0.37037086],
 'std': [0.33216846, 0.33216846]},
 {'age': 31.0,
 'mean': [0.52112424, 0.4788758],
 'std': [0.3004837, 0.30048367]},
 {'age': 33.0,
 'mean': [0.61710674, 0.38289332],
 'std': [0.3396127, 0.33961272]},
 {'age': 34.98275862068965,
 'mean': [0.61710674, 0.38289332],
 'std': [0.3396127, 0.33961272]},
 {'age': 36.0,
 'mean': [0.62249655, 0.37750348],
 'std': [0.33859333, 0.33859333]},
 {'age': 38.0,
 'mean': [0.6321857, 0.36781433],
 'std': [0.34262648, 0.34262648]},
 {'age': 40.0, 'mean': [0.6353405, 0.3646595], 'std': [0.3420124, 0.3420124]},
 {'age': 43.0,
 'mean': [0.6327028, 0.36729714],
 'std': [0.33809677, 0.33809677]},
 {'age': 45.0,
 'mean': [0.6343053, 0.36569482],
 'std': [0.33775553, 0.33775553]},
 {'age': 49.0,
 'mean': [0.5965565, 0.40344357],
 'std': [0.3381666, 0.3381666]},
 {'age': 54.0,
 'mean': [0.64475715, 0.35524285],
 'std': [0.3373284, 0.3373284]},
 {'age': 60.0,
 'mean': [0.6563791, 0.34362087],
 'std': [0.34226355, 0.34226355]},
 {'age': 80.0,
 'mean': [0.6601273, 0.33987272],
 'std': [0.34131092, 0.34131092]}]}

```

```
# The explanation can also be returned as Pandas.DataFrame with  
pdp_age.as_dataframe()
```

	age	mean_False	std_False	mean_True	std_True
0	0.166700	0.219623	0.186594	0.780377	0.186594
1	3.000000	0.195860	0.201552	0.804140	0.201552
2	9.000000	0.213872	0.194057	0.786128	0.194057
3	14.172414	0.519542	0.313260	0.480458	0.313260
4	17.000000	0.665386	0.323477	0.334614	0.323477
5	18.000000	0.665386	0.323477	0.334614	0.323477
6	19.000000	0.603867	0.329272	0.396133	0.329272
7	20.000000	0.590466	0.324003	0.409534	0.324003
8	21.000000	0.594511	0.325571	0.405489	0.325571
9	22.000000	0.570493	0.324753	0.429507	0.324753
10	23.000000	0.568903	0.325895	0.431097	0.325895
11	24.000000	0.572269	0.324574	0.427731	0.324574
12	25.000000	0.564627	0.321411	0.435373	0.321411
13	26.000000	0.563571	0.320828	0.436429	0.320828
14	27.000000	0.581065	0.317451	0.418935	0.317451
15	28.000000	0.626350	0.329872	0.373650	0.329872
16	29.000000	0.623764	0.330370	0.376236	0.330370
17	30.000000	0.629629	0.332168	0.370371	0.332168
18	31.000000	0.521124	0.300484	0.478876	0.300484
19	33.000000	0.617107	0.339613	0.382893	0.339613
20	34.982759	0.617107	0.339613	0.382893	0.339613
21	36.000000	0.622497	0.338593	0.377503	0.338593
22	38.000000	0.632186	0.342626	0.367814	0.342626
23	40.000000	0.635341	0.342012	0.364659	0.342012
24	43.000000	0.632703	0.338097	0.367297	0.338097
25	45.000000	0.634305	0.337756	0.365695	0.337756
26	49.000000	0.596556	0.338167	0.403444	0.338167
27	54.000000	0.644757	0.337328	0.355243	0.337328
28	60.000000	0.656379	0.342264	0.343621	0.342264
29	80.000000	0.660127	0.341311	0.339873	0.341311

### 11.5.2.5 References

- [Partial Dependence Plot](#)
- [Vanderbilt Biostatistics - titanic data](#)

## 11.5.3 Feature Importance Explanations

### 11.5.3.1 Overview

Feature permutation importance is a model-agnostic global explanation method that provides insights into a machine learning model's behavior. It estimates and ranks feature importance based on the impact each feature has on the trained machine learning model's predictions.

### 11.5.3.2 Description

Feature permutation importance measures the predictive value of a feature for any black box estimator, classifier, or regressor. It does this by evaluating how the prediction error increases when a feature is not available. Any scoring metric can be used to measure the prediction error. For example,  $F_1$  for classification or  $R^2$  for regression. To avoid actually removing features and retraining the estimator for each feature, the algorithm randomly shuffles the feature values effectively adding noise to the feature. Then, the prediction error of the new dataset is compared with the prediction error of the original dataset. If the model heavily relies on the column being shuffled to accurately predict the target variable, this random re-ordering causes less accurate predictions. If the model does not rely on the feature for its predictions, the prediction error remains unchanged.

The following summarizes the main steps in computing feature permutation importance explanations:

- Start with a trained machine learning model.
- Calculate the baseline prediction error on the given dataset. For example, train dataset or test dataset.
- For each feature:
  1. Randomly shuffle the feature column in the given dataset.
  2. Calculate the prediction error on the shuffled dataset.
  3. Store the difference between the baseline score and the shuffled dataset score as the feature importance. For example, baseline score - shuffled score.
- Repeat the preceding three steps multiple times then report the average. Averaging mitigates the effects of random shuffling.
- Rank the features based on the average impact each feature has on the model's score. Features that have a larger impact on the score when shuffled are assigned higher importance than features with minimal impact on the model's score.
- In some cases, randomly permuting an unimportant feature can actually have a positive effect on the model's prediction so the feature's contribution to the model's predictions is effectively noise. In the feature permutation importance visualizations, ADS caps any negative feature importance values at zero.

### 11.5.3.3 Interpretation

Feature permutation importance explanations generate an ordered list of features along with their importance values. Interpreting the output of this algorithm is straightforward. Features located at higher ranks have more impact on the model predictions. Features at lower ranks have less impact on the model predictions. Additionally, the importance values represent the relative importance of features.

The output supports three types of visualizations. They are all based on the same data but present the data differently for various use cases:

- **Bar chart** ('bar'): The bar chart shows the model's view of the relative feature importance. The x-axis highlights feature importance. A longer bar indicates higher importance than a shorter bar. Each bar also shows the average feature importance value along with the standard deviation of importance values across all iterations of the algorithm (mean importance +/- standard deviation\*). Negative importance values are capped at zero. The y-axis shows the different features in the relative importance order. The top being the most important, and the bottom being the least important.
- **Box plot** ('box\_plot'): The detailed box plot shows the feature importance values across the iterations of the algorithm. These values are used to compute the average feature importance and the corresponding standard deviations shown in the bar chart. The x-axis shows the impact that permuting a given feature had on the model's prediction score. The y-axis shows the different features in the relative importance order. The top being the most important, and the bottom being the least important. The minimum, first quartile, median, third quartile, and a maximum of the feature importance values across different iterations of the algorithm are shown by each box.
- **Detailed scatter plot** ('detailed'): The detailed bar chart shows the feature importance values for each iteration of the algorithm. These values are used to compute the average feature importance values and the corresponding standard deviations shown in the bar chart. The x-axis shows the impact that permuting a given feature had on the model's prediction score. The y-axis shows the different features in the relative importance order. The top being the most important, and the bottom being the least important. The color of each dot in the graph indicates the quality of the permutation for this iteration, which is computed by measuring the correlation of the permuted feature column relative to the original feature column. For example, how different is the permuted feature column versus the original feature column.

### 11.5.3.4 Examples

This example generates and visualizes a global feature permutation importance explanation on the [Titanic dataset](#). The model is constructed using the ADS OracleAutoMLProvider. However, the ADS model explainers work with any model (classifier or regressor) that is wrapped in an ADSModel object.

```
import logging
import requests

from ads.automl.driver import AutoML
from ads.automl.provider import OracleAutoMLProvider
from ads.dataset.factory import DatasetFactory
from os import path

# Prepare and load the dataset
titanic_data_file = '/tmp/titanic.csv'
if not path.exists(titanic_data_file):
    # fetch and save some data
    print('fetching data from web...', end=" ")
    # Data source: https://www.openml.org/d/40945
    r = requests.get('https://www.openml.org/data/get_csv/16826755/phpMYEkM1')
    with open(titanic_data_file, 'wb') as fd:
```

(continues on next page)

(continued from previous page)

```

        fd.write(r.content)
    print("Done")
ds = DatasetFactory.open(
    titanic_data_file, target="survived").set_positive_class(True)
ds = ds.drop_columns(['name', 'ticket', 'cabin', 'boat',
                     'body', 'home.dest'])
ds = ds[ds['age'] != '?'].astype({'age': 'float64'})
ds = ds[ds['fare'] != '?'].astype({'fare': 'float64'})
train, test = ds.train_test_split(test_size=0.2)

# Build the model using AutoML. 'model' is a subclass of type ADSModel.
# Note that the ADSExplainer below works with any model (classifier or
# regressor) that is wrapped in an ADSModel
ml_engine = OracleAutoMLProvider(n_jobs=-1, loglevel=logging.ERROR)
oracle_automl = AutoML(train, provider=ml_engine)
model, baseline = oracle_automl.train()

# Create the ADS explainer object, which is used to construct global
# and local explanation objects. The ADSExplainer takes as input the
# model to explain and the train/test dataset
from ads.explanations.explainer import ADSExplainer
explainer = ADSExplainer(test, model, training_data=train)

# With ADSExplainer, create a global explanation object using
# the MLXGlobalExplainer provider
from ads.explanations.mlx_global_explainer import MLXGlobalExplainer
global_explainer = explainer.global_explanation(
    provider=MLXGlobalExplainer())

# A summary of the global feature permutation importance algorithm and
# how to interpret the output can be displayed with
global_explainer.feature_importance_summary()

# Compute the global Feature Permutation Importance explanation
importances = global_explainer.compute_feature_importance()

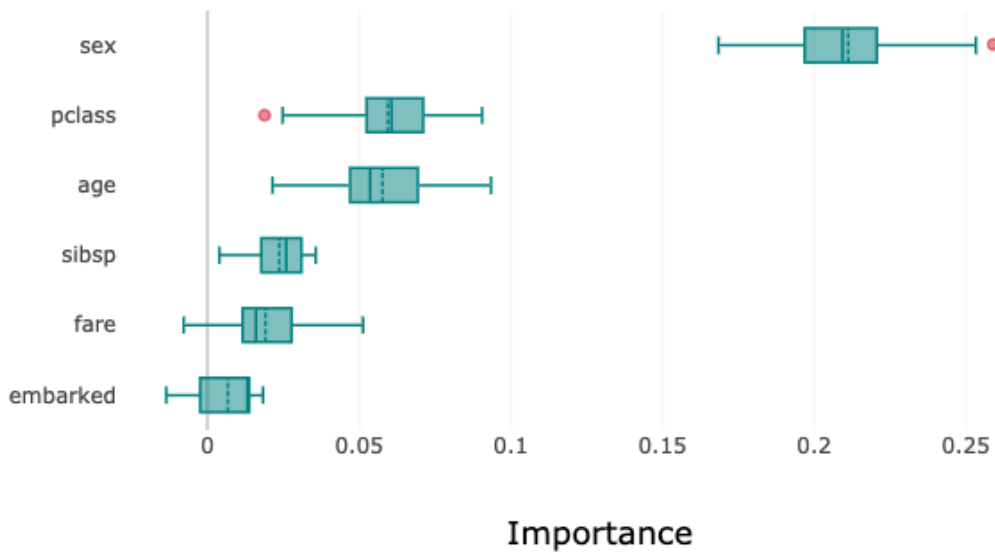
# ADS supports multiple visualizations for the global Feature
# Permutation Importance explanations (see "Interpretation" above)

# Simple bar chart highlighting the average impact on model score
# across multiple iterations of the algorithm
importances.show_in_notebook()

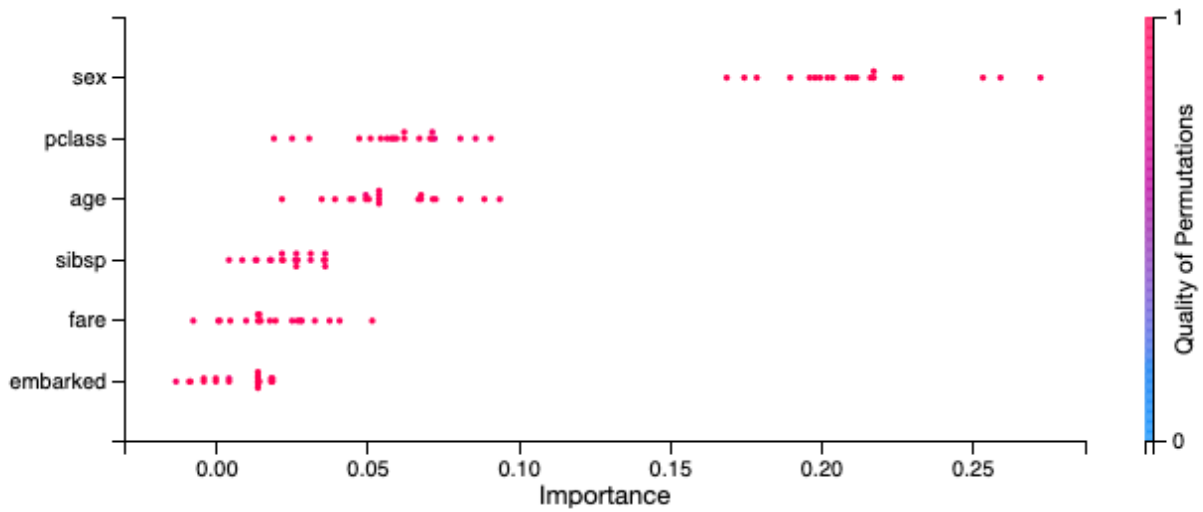
```



```
# Box plot highlighting the mean, median, quartiles, and min/max
# impact on model score across multiple iterations of the algorithm
importances.show_in_notebook('box_plot')
```



```
# Detailed scatter plot highlighting the individual impacts on
# model score across multiple iterations of the algorithm
importances.show_in_notebook('detailed')
```



```
# The raw explanaiton data used to generate the visualizations, as well  
# as the runtime performance information can be extracted with  
importances.get_diagnostics()
```



```
{'explanations': [{ 'feature': 'sex',
  'attribution': 0.21124298808944758,
  'attribution_std': 0.02617818201628649,
  'confidence': 0.9929626508892098,
  'confidence_std': 0.016704326572772182},
{ 'feature': 'pclass',
  'attribution': 0.059492724602767874,
  'attribution_std': 0.018261289784839586,
  'confidence': 0.9421497725417008,
  'confidence_std': 0.04336997475992779},
{ 'feature': 'age',
  'attribution': 0.057728878073588355,
  'attribution_std': 0.017633783394690756,
  'confidence': 0.9606087248537752,
  'confidence_std': 0.031929775401309375},
{ 'feature': 'sibsp',
  'attribution': 0.02371181564197248,
  'attribution_std': 0.009087998301193395,
  'confidence': 0.9422402486313869,
  'confidence_std': 0.03738023968977173},
{ 'feature': 'fare',
  'attribution': 0.019121222158654673,
  'attribution_std': 0.014307567871540862,
  'confidence': 0.9597909343483199,
  'confidence_std': 0.025478489355540486},
{ 'feature': 'embarked',
  'attribution': 0.006731802664474656,
  'attribution_std': 0.010294686196767218,
  'confidence': 0.9824752088136544,
  'confidence_std': 0.03584913237068881}],
'explanations_stats': { 'n_iterations': 20,
  'total_runtime': 8.949006080627441,
  'iteration_average_runtime': 0.44195606708526614}}
```

### 11.5.3.5 References

- [Feature importance](#)
- [Perutation importance](#)
- [Vanderbilt Biostatistics - titanic data](#)

## 11.5.4 Enhanced LIME

### 11.5.4.1 Overview

Local explanations target specific predictions from the machine learning model. The goal is to understand why the model made a particular prediction.

There are multiple different forms of local explanations, such as feature attribution explanations and exemplar-based explanations. ADS supports local feature attribution explanations. They help to identify the most important features leading towards a given prediction.

While a given feature might be important for the model in general, the values in a particular sample may cause certain features to have a larger impact on the model's predictions than others. Furthermore, given the feature values in a specific sample, local explanations can also estimate the contribution that each feature had towards or against a target prediction. For example, does the current value of the feature have a positive or negative effect on the prediction probability of the target class? Does the feature increase or decrease the predicted regression target value?

The Enhanced Local Interpretable Model-Agnostic Explanation (LIME) is a model-agnostic local explanation method. It provides insights into why a machine learning model made a specific prediction.

### 11.5.4.2 Description

ADS provides an enhanced version of Local Interpretable Model-Agnostic Explanations (LIME), which improves on the explanation quality, performance, and interpretability. The key idea behind LIME is that while the global behavior of a machine learning model might be very complex, the local behavior may be much simpler. In ADS, local refers to the behavior of the model on similar samples. LIME tries to approximate the local behavior of the complex machine learning model through the use of a simple, inherently interpretable surrogate model. For example, a linear model. If the surrogate model is able to accurately approximate the complex model's local behavior, ADS can generate a local explanation of the complex model from the interpretable surrogate model. For example, when data is centered and scaled the magnitude and sign of the coefficients in a linear model indicate the contribution each feature has towards the target variable.

The predictions from complex machine learning models are challenging to explain and are generally considered as a black box. As such, ADS refers to the model to be explained as the black box model. ADS supports classification and regression models on tabular or text-based datasets (containing a single text-based feature).

The main steps in computing a local explanation for tabular datasets are:

- Start with a trained machine learning model (the black box model).
- Select a specific sample to explain ( $x_{\text{exp}}$ ).
- Randomly generate a large sample space in a nearby neighborhood around  $x_{\text{exp}}$ . The sample space is generated based on the feature distributions from the training dataset. Each sample is then weighted based on its distance from  $x_{\text{exp}}$  to give higher weight to samples that are closer to  $x_{\text{exp}}$ . ADS provides several enhancements, over the standard algorithm, to improve the quality and locality of the sample generation and weighting methods.
- Using the black box model, generate a prediction for each of the randomly generated local samples. For classification tasks, compute the prediction probabilities using `predict_proba()`. For regression tasks, compute the predicted regression value using `predict()`.
- Fit a linear surrogate model on the predicted values from the black box model on the local generated sample space. If the surrogate model is able to accurately match the output of the black box model (referred to as surrogate model fidelity), the surrogate model can act as a proxy for explaining the local behavior of the black box model. For classification tasks, the surrogate model is a linear regression model fit on the prediction probabilities of the black box model. Consequently, for multinomial classification tasks, a separate surrogate model is required to explain each class. In that case, the explanation indicates if a feature contributes towards the specified class or

against the specified class (for example, towards one of the other  $N$  classes). For regression tasks, the surrogate model is a linear regression model fit on the predicted regression values from the black box model.

- There are two available techniques for fitting the surrogate model:

- Use the features directly:

The raw (normalized) feature values are used to fit the linear surrogate model directly. This results in a normal linear model. A positive coefficient indicates that when the feature value increases, the target variable increases. A negative coefficient indicates that when a feature value increases, the target variable decreases. Categorical features are converted to binary values. A value of 1 indicates that the feature in the generated sample has the same value as  $x_{\text{exp}}$  and a value of 0 indicates that the feature in the generated sample has a different value than  $x_{\text{exp}}$ .

- Translate the features to an interpretable feature space:

Continuous features are converted to categorical features by discretizing the feature values (for example, quartiles, deciles, and entropy-based). Then, all features are converted to binary values. A value of 1 indicates that the feature in the generated sample has the same value as  $x_{\text{exp}}$  (for example, the same categorical value or the continuous feature falls in the same bin) and a value of 0 indicates that the feature in the generated sample has a different value than  $x_{\text{exp}}$  (for example, a different categorical value or the continuous feature falls in a different bin). The interpretation of the linear model here is a bit different from the regression model. A positive coefficient indicates that when a feature has the same value as  $x_{\text{exp}}$  (for example, the same category), the feature increased the prediction output from the black box model. Similarly, negative coefficients indicate that when a feature has the same value as  $x_{\text{exp}}$ , the feature decreased the prediction output from the black box model. This does not say what happens when the feature is in a different category than  $x_{\text{exp}}$ . It only provides information when the specific feature has the same value as  $x_{\text{exp}}$  and if it positively or negatively impacts the black box model's prediction.

- The explanation is an ordered list of feature importances extracted from the coefficients of the linear surrogate model. The magnitude of the coefficients indicates the relative feature importance and the sign indicates whether the feature has a positive or negative impact on the black box model's prediction.
- The algorithm is similar to text-based datasets. The main difference is in the random local sample space generation. Instead of randomly generating samples based on the feature distributions, a large number of local samples are generated by randomly removing subsets of words from the text sample. Each of the randomly generated samples is converted to a binary vector-based on the existence of a word. For example, the original sample to explain,  $x_{\text{exp}}$ , contains 1s for every word. If the randomly generated sample has the same word as  $x_{\text{exp}}$ , it is a value of 1. If the word has been removed in the randomly generated sample, it is a value of 0. In this case, the linear surrogate model evaluates the behavior of the model when the word is there or not.

Additionally, an upper bound can be set on the number of features to include in the explanation (for example, explain the top- $N$  most important features). If the specified number of features is less than the total number of features, a simple feature selection method is applied prior to fitting the linear surrogate model. The black box model is still evaluated on all features, but the surrogate model is only fits on the subset of features.

#### 11.5.4.3 Interpretation

ADS provides multiple enhancements to the local visualizations from LIME. The explanation is presented as a grid containing information about the black box model, information about the local explainer, and the actual local explanation. Each row in the grid is described as:

- Model (first row)
  - The left column presents information about the black box model and the model's prediction. For example, the type of the black box model, the true label/value for the selected sample to explain, the predicted value from the black box model, and the prediction probabilities (classification) or prediction values (regression).

- The right column displays the sample to explain. For tabular datasets, this is a table showing the feature names and corresponding values for this sample. For text datasets, this shows the text sample to explain.
- Explainer (second row)
  - The left column presents the explainer configuration parameters, such as the underlying local explanation algorithm used (for example, LIME), the type of surrogate model (for example, linear), the number of randomly generated local samples (for example, 5000) to train the local surrogate model ( $N_t$ ), whether continuous features were discretized or not.
  - The right column provides a legend describing how to interpret the model explanations.
- Explanations (remaining rows)
  - For classification tasks, a local explanation can be generated for each of the target labels (since the surrogate model is fit to the prediction probabilities from the black box model). For binary classification, the explanation for one class will mirror the other. For multinomial classification, the explanations describe how each feature contributes towards or against the specified target class. If the feature contributes against the specified target class (for example, decreases the prediction probability), it increases the prediction probability of one or more other target classes. The explanation for each target class is shown as a separate row in the Explanation section.
  - The Feature Importances section presents the actual local explanation. The explanation is visualized as a horizontal bar chart of feature importance values, ordered by relative feature importance. Features with larger bars (top) are more important than features with shorter bars (bottom). Positive feature importance values (to the right) indicate that the feature increases the prediction target value. Negative feature importance values (to the left) indicate that the feature decreases the prediction target value. Depending on whether continuous features are discretized or not changes the interpretation of this value (for example, whether the specific feature value indicates a positive/negative attribution, or whether an increase/decrease in the feature value indicates a positive/negative attribution). If the features are discretized, the corresponding range is included. The feature importance value is shown beside each bar. This can either be the raw coefficient taken from the linear surrogate model or can be normalized such that all importance values sum to one. For text datasets, the explanation is visualized as a word cloud. Important words that have a large positive contribution towards a given prediction (for example, increase the prediction value) are shown larger than unimportant words that have a less positive impact on the target prediction.
- The Explanation Quality section presents information about the quality of the explanation. It is further broken down into two sections:
  - Sample Distance Distributions

This section presents the sample distributions used to train ( $N_t$ ) and evaluate ( $N_{v_{\#}}$ ) the local surrogate model based on the distances (Euclidean) of the generated samples from the sample to explain. This highlights the locality of generated sample spaces where the surrogate model (explainer) is trained and evaluated. The distance distribution from the sample to explain for the actual dataset used to train the black box model, Train, is also shown. This highlights the locality of  $N_t$  relative to the entire train dataset. For the generated evaluation sample spaces ( $N_{v_{\#}}$ ), the sample space is generated based on a percentile value of the distances in Train relative to the sample to explain. For example,  $N_{v_4}$  is generated with the maximum distance being limited to the 4<sup>th</sup> percentile of the distances in train from the sample to explain.
  - Evaluation Metrics

This section presents the fidelity of the surrogate model relative to the black box model on the randomly generated sample spaces used to fit and evaluate the surrogate model. In other words, this section evaluates how accurately the surrogate model approximates the local behavior of the complex black box model. Multiple different regression and classification metrics are supported. For classification tasks, ADS supports both regression and classification metrics. Regression metrics are computed on the raw prediction probabilities between the surrogate model and the black box model. For classification metrics, the prediction probabilities are converted to the corresponding target labels and are compared between the surrogate model

and the black box model. Explanations for regression tasks only support regression metrics. Supported regression metrics: MSE, RMSE (default),  $R^2$ , MAPE, SMAPE, Two-Sample Kolmogorov-Smirnov Test, Pearson Correlation (default), and Spearman Correlation. Supported classification metrics:  $F_1$ , Accuracy, Recall, and ROC\_AUC.

- Performance

Explanation time in seconds.

#### 11.5.4.4 Example

This example generates and visualizes local explanations on the `Titanic` dataset. The model is constructed using the ADS `OracleAutoMLProvider`. However, the ADS model explainers work with any model (classifier or regressor) that is wrapped in an `ADSModel` object.

```
import logging
import requests

from ads.automl.driver import AutoML
from ads.automl.provider import OracleAutoMLProvider
from ads.dataset.factory import DatasetFactory
from os import path

# Prepare and load the dataset
titanic_data_file = '/tmp/titanic.csv'
if not path.exists(titanic_data_file):
    # fetch and save some data
    print('fetching data from web...', end=" ")
    # Data source: https://www.openml.org/d/40945
    r = requests.get('https://www.openml.org/data/get_csv/16826755/phpMYEkM1')
    with open(titanic_data_file, 'wb') as fd:
        fd.write(r.content)
    print("Done")
ds = DatasetFactory.open(
    titanic_data_file, target="survived").set_positive_class(True)
ds = ds.drop_columns(['name', 'ticket', 'cabin', 'boat',
                     'body', 'home.dest'])
ds = ds[ds['age'] != '?'].astype({'age': 'float64'})
ds = ds[ds['fare'] != '?'].astype({'fare': 'float64'})
train, test = ds.train_test_split(test_size=0.2)

# Build the model using AutoML. 'model' is a subclass of type ADSModel.
# Note that the ADSExplainer below works with any model (classifier or
# regressor) that is wrapped in an ADSModel
ml_engine = OracleAutoMLProvider(n_jobs=-1, loglevel=logging.ERROR)
oracle_automl = AutoML(train, provider=ml_engine)
model, baseline = oracle_automl.train()

# Create the ADS explainer object, which is used to construct
# global and local explanation objects. The ADSExplainer takes
# as input the model to explain and the train/test dataset
from ads.explanations.explainer import ADSExplainer
explainer = ADSExplainer(test, model, training_data=train)
```

(continues on next page)

(continued from previous page)

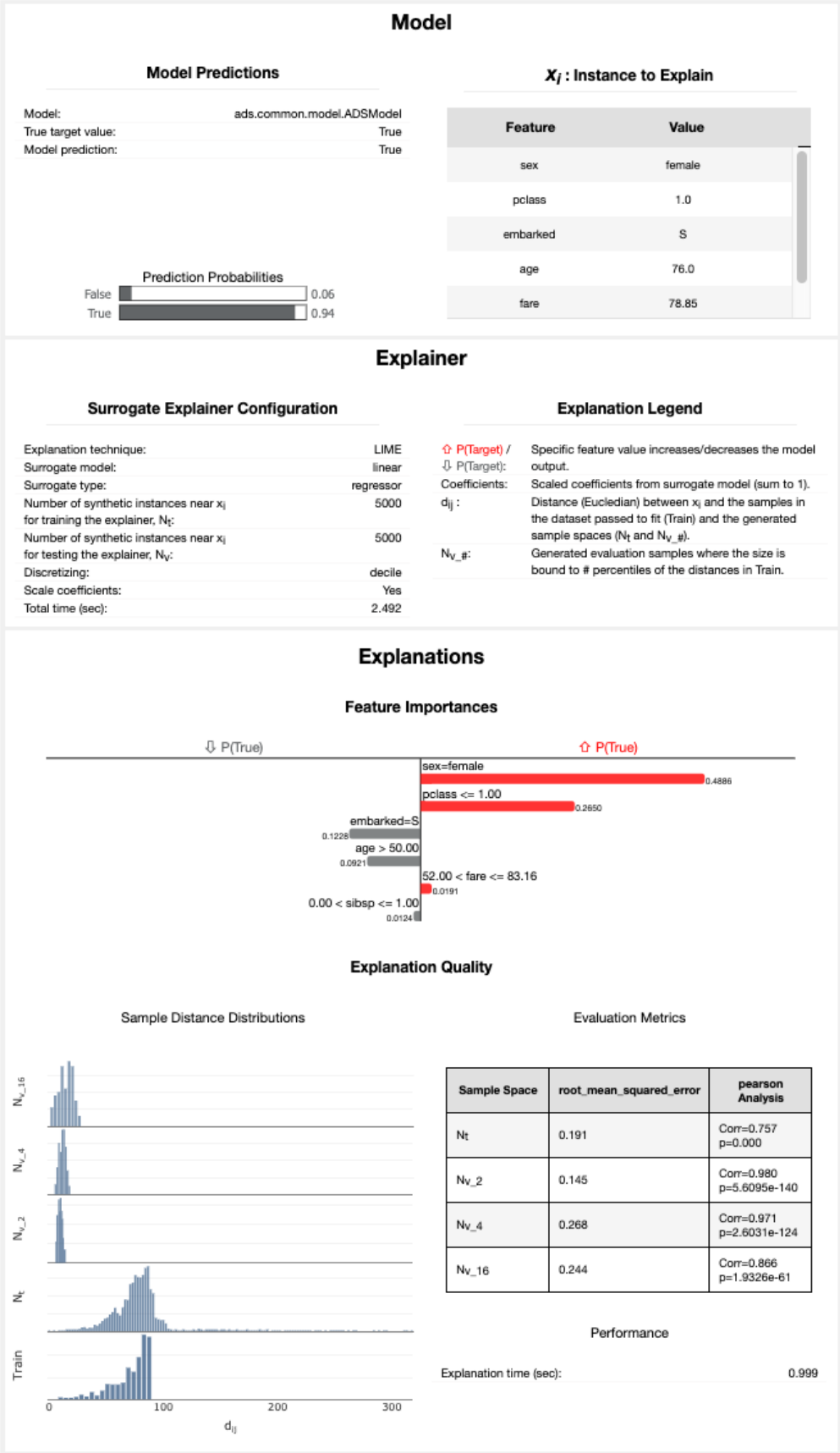
```
# With ADSExplainer, create a local explanation object using
# the MLXLocalExplainer provider
from ads.explanations.mlx_local_explainer import MLXLocalExplainer
local_explainer = explainer.local_explanation(
    provider=MLXLocalExplainer())

# A summary of the local explanation algorithm and how to interpret
# the output can be displayed with
local_explainer.summary()

# Select a specific sample (instance/row) to generate a local
# explanation for
sample = 13

# Compute the local explanation on our sample from the test set
explanation = local_explainer.explain(test.X.iloc[sample:sample+1],
                                     test.y.iloc[sample:sample+1])

# Visualize the explanation for the label True (Survived). See
# the "Interpretation" section above for more information
explanation.show_in_notebook(labels=True)
```



```
# The raw explanaiton data used to generate the visualizations, as well  
# as the runtime performance information can be extracted with  
explanation.get_diagnostics()
```



```

('explanations': {0: [{'feature': 'sex',
  'attribution': -0.4886071283440021,
  'value': 'female',
  'line_feature_str': 'sex=female',
  'inv_coef': -0.4052640255390418},
{'feature': 'pclass',
  'attribution': -0.2650143690243627,
  'value': 1.0,
  'line_feature_str': 'pclass <= 1.00',
  'inv_coef': -0.2198101169348623},
{'feature': 'embarked',
  'attribution': 0.12281563773925595,
  'value': 'S',
  'line_feature_str': 'embarked=S',
  'inv_coef': 0.10186662629758687},
{'feature': 'age',
  'attribution': 0.09209659437480645,
  'value': 76.0,
  'line_feature_str': 'age > 50.00',
  'inv_coef': 0.07638741723082866},
{'feature': 'fare',
  'attribution': -0.019091348887562875,
  'value': 78.85,
  'line_feature_str': '52.00 < fare <= 83.16',
  'inv_coef': -0.01583488339469498},
{'feature': 'sibsp',
  'attribution': 0.01237492163000979,
  'value': 1.0,
  'line_feature_str': '0.00 < sibsp <= 1.00',
  'inv_coef': 0.010264096171714174}],
1: [{'feature': 'sex',
  'attribution': 0.4886071289258118,
  'value': 'female',
  'line_feature_str': 'sex=female',
  'inv_coef': 0.40526402582620324},
{'feature': 'pclass',
  'attribution': 0.2650143690673043,
  'value': 1.0,
  'line_feature_str': 'pclass <= 1.00',
  'inv_coef': 0.21981011686449275},
{'feature': 'embarked',
  'attribution': -0.12281563786873055,
  'value': 'S',
  'line_feature_str': 'embarked=S',
  'inv_coef': -0.10186662635585929},
{'feature': 'age',
  'attribution': -0.09209659370130899,
  'value': 76.0,
  'line_feature_str': 'age > 50.00',
  'inv_coef': -0.07638741663537962},
{'feature': 'fare',
  'attribution': 0.01909134822337224,
  'value': 78.85,
  'line_feature_str': '52.00 < fare <= 83.16',
  'inv_coef': 0.01583488283616207},
{'feature': 'sibsp',
  'attribution': -0.012374922213471952,
  'value': 1.0,
  'line_feature_str': '0.00 < sibsp <= 1.00',
  'inv_coef': -0.0102640966070472}],
'explanation_stats': {0: {'runtime': 1.0013508796691895,
  'train_report': {'root_mean_squared_error': 0.1913996858701819,
  'pearson': (0.7571939470713043, 0.0)},
  'validation_report': {'root_mean_squared_error': 0.14885348708768853,
  'pearson': (0.980253326704337, 2.4597276552456752e-141),
  'distance': (array([17, 26, 23, 26, 32, 34, 19, 17, 6]),
  array([6.16250218, 0.0884351, 7.89577612, 11.0489053, 9.86623395,
  10.79216691, 11.71809986, 12.64403281, 13.56996576, 14.49589871])),
  4: {'root_mean_squared_error': 0.26391348642032847,
  'pearson': (0.977143707727337, 4.57053549877471e-126),
  'distance': (array([10, 13, 15, 30, 35, 34, 28, 23, 12]),
  array([5.30845668, 6.7454686, 8.18248051, 9.61949243, 11.05650434,
  12.49351625, 13.93052817, 15.36754008, 16.804552, 18.24156391])),
  16: {'root_mean_squared_error': 0.252222562007294,
  'pearson': (0.8451737610940484, 7.30606857254383e-55),
  'distance': (array([7, 15, 18, 30, 37, 34, 37, 17, 5]),
  array([1.48652616, 4.74265194, 7.89577612, 11.0489053, 14.20202449,
  17.35514867, 20.50827285, 23.66139704, 26.81452122, 29.9676454])),
  1: {'runtime': 0.999420597839355,
  'train_report': {'root_mean_squared_error': 0.191399685806725,
  'pearson': (0.7571939470713043, 0.0)},
  'validation_report': {'root_mean_squared_error': 0.1454804921366557,
  'pearson': (0.9796132002600891, 5.609492972343477e-140),
  'distance': (array([11, 25, 13, 27, 34, 27, 38, 35, 7]),
  array([6.53755597, 7.41449311, 8.29143026, 9.1683674, 10.04530454,
  10.92224168, 11.79917883, 12.67611597, 13.55305311, 14.42999025])),
  4: {'root_mean_squared_error': 0.26782522120554206,
  'pearson': (0.9705180239448721, 2.6030634638264783e-124),
  'distance': (array([6, 16, 30, 25, 38, 38, 28, 14, 5]),
  array([5.6595933, 7.12347081, 8.29143026, 9.1683674, 10.04530454,
  11.79917883, 12.67611597, 13.55305311, 14.42999025])),
  16: {'root_mean_squared_error': 0.24440748498897852,
  'pearson': (0.865772883218871, 1.9325796113627693e-61),
  'distance': (array([11, 18, 20, 22, 38, 35, 15, 6]),
  array([2.36948461, 5.38330542, 8.39712622, 11.41094703, 14.42476783,
  17.43858864, 20.45240944, 23.46623025, 26.48005105, 29.49387186])),
  'runtime': 2.492041826481619,
  'training_distances': (array([5, 3, 4, 6, 11, 7, 17, 9, 20, 35, 34,
  34, 38,
  72, 63, 96, 147, 142]),
  array([9.98110715, 14.54846188, 19.1158166, 23.68317132, 28.25052605,
  32.81788077, 37.38523549, 41.95259022, 46.51994494, 51.08729967,
  55.65465439, 60.22200911, 64.78916384, 69.35671856, 73.92407328,
  78.49142801, 83.05878272, 87.62613745, 92.19349218])),
  'nt_distances': (array([1, 0, 1, 0, 1, 3, 3, 7, 5, 7, 10, 5, 1
  2,
  21, 23, 15, 19, 18, 36, 31, 42, 57, 70, 76, 94, 108,
  132, 102, 89, 134, 179, 176, 266, 274, 317, 306, 303, 318, 357,
  367, 237, 214, 75, 67, 62, 63, 44, 14, 9, 3, 7, 2,
  2, 6, 4, 2, 4, 9, 4, 13, 6, 4, 6, 6, 7,
  6, 7, 4, 9, 5, 3, 8, 3, 9, 3, 3, 1,
  5, 2, 2, 3, 3, 5, 3, 4, 8, 0, 3, 3, 3,
  2, 0, 1, 2, 3, 1, 2, 2, 1, 3, 1, 2, 1,
  0, 1, 2, 1, 0, 2, 1, 2, 1, 2, 1, 1,
  0, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0, 1, 0,
  1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
  1]),
  array([0.2206455, 2.44412914, 6.6619371,
  8.88258279, 11.10322849, 13.32387419, 15.54451989,
  17.76516559, 19.98581129, 22.20645699, 24.42710268,
  26.64774838, 28.86839408, 31.08903978, 33.30968548,
  35.53031118, 37.75097688, 39.97162257, 42.19226827,
  44.41291397, 46.63355967, 48.85420537, 51.07485107,
  53.29549677, 55.51614246, 57.73678816, 59.95743386,
  62.17807956, 64.39872526, 66.61937096, 68.84001666,
  71.06066235, 73.28130805, 75.50195375, 77.72259945,
  79.94324515, 82.16389085, 84.38453655, 86.60518225,
  88.82582794, 91.04647164, 93.26711934, 95.48776504,
  97.70841074, 99.92905644, 102.14970214, 104.37034783,
  106.59099353, 108.81163923, 111.03228493, 113.25293063,
  115.47357633, 117.69422203, 119.91486772, 122.13551342,
  124.35615919, 126.57680482, 128.79745052, 131.01809622,
  133.23874192, 135.45938761, 137.68003331, 139.90067901,
  142.12132471, 144.34197041, 146.56261611, 148.78326181,
  151.0039075, 153.2245532, 155.4451989, 157.6658446,
  159.8864903, 162.107136, 164.3277817, 166.54842739,
  168.76907309, 170.98971879, 173.21036449, 175.43101019,
  177.65165589, 179.87230159, 182.09294728, 184.31359298,
  186.53423868, 188.75488438, 190.97553008, 193.19617578,
  195.41682148, 197.63746717, 199.85811287, 202.07875857,
  204.29940427, 206.52004997, 208.74069567, 210.96134137,
  213.18198707, 215.40263276, 217.62327846, 219.84392416,
  222.06456986, 224.28521556, 226.50586126, 228.72650695,
  230.94715265, 233.16779835, 235.38844405, 237.60908975,
  239.82971545, 242.05038115, 244.27102685, 246.49167254,
  248.71231824, 250.93296394, 253.15360964, 255.37425534,
  257.59490104, 259.81554674, 262.03619243, 264.25683813,
  266.47748383, 268.69812953, 270.91877523, 273.13942093,
  275.36006663, 277.58071332, 279.80135802, 282.02200372,
  284.24264942, 286.46329512, 288.68394082, 290.90458652,
  293.12523221, 295.34587791, 297.56652361, 299.78716931,
  302.00781501, 304.22846071, 306.44910641, 308.66975221,
  310.89039378, 313.1110435, 315.3316892, 317.55233449,
  319.7729806 ]))})

```

### 11.5.4.5 References

- [LIME](#)
- [Vanderbilt Biostatistics - titanic data](#)
- [Why Should I Trust You? Explaining the Predictions of Any Classifier](#)

## 11.5.5 WhatIf Explainer

### 11.5.5.1 Description

The WhatIf explainer tool helps to understand how changes in an observation affect a model's prediction. Use it to explore a model's behavior on a single observation or the entire dataset by asking “what if” questions.

The WhatIf explainer has the following methods:

- `explore_predictions`: Explore the relationship between feature values and the model predictions.
- `explore_sample`: Modify the values in an observation and see how the prediction changes.

### 11.5.5.2 Example

In this example, a WhatIf explainer is created, and then the `explore_predictions()`, and `explore_sample()` methods are demonstrated. A tree-based model is used to make predictions on the Boston housing dataset.

```
from ads.common.model import ADSModel
from ads.dataset.dataset_browser import DatasetBrowser
from ads.dataset.label_encoder import DataFrameLabelEncoder
from ads.explanations.explainer import ADSExplainer
from ads.explanations.mlx_whatif_explainer import MLXWhatIfExplainer
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import LabelEncoder
import logging
import warnings

logging.basicConfig(format='%(levelname)s:%(message)s', level=logging.ERROR)
warnings.filterwarnings('ignore')

ds = DatasetBrowser.sklearn().open("boston").set_target("target")
train, test = ds.train_test_split(test_size=0.2)

X_boston = train.X.copy()
y_boston = train.y.copy()

le = DataFrameLabelEncoder()
X_boston = le.fit_transform(X_boston)

# Model Training
ensemble_regressor = ExtraTreesRegressor(n_estimators=245, random_state=42)
ensemble_regressor.fit(X_boston, y_boston)
model = ADSModel.from_estimator(make_pipeline(le, ensemble_regressor), name=
↪ "ExtraTreesRegressor")
```

(continues on next page)

(continued from previous page)

```
# Build a WhatIf Explainer
explainer = ADSExplainer(test, model, training_data=train)
whatif_explainer = explainer.whatif_explanation(provider=MLXWhatIfExplainer())
```

The Sample Explorer method, `explore_sample()`, opens a GUI that has a single observation. The values of that sample can then be changed. By clicking **Run Inference**, the model computes the prediction with the updated feature values. The interface shows the original values and the values that have been changed.

`example_sample()` accepts the `row_idx` parameter that specifies the index of the observation that is to be evaluated. The default is zero (0). The `features` parameter lists the feature names that are shown in the interface. By default, it displays all features. For datasets with a large number of features, this can be cumbersome so the `max_features` parameter can be used to display only the first *n* features.

The following command opens the Sample Explorer. Change the values then click **Run Inference** to see how the prediction changes.

```
whatif_explainer.explore_sample()
```

### Select and Explore Sample

#### Row Selection

Select a sample between 0 and 101

Row Index:

Select Sample

#### Sample (Row: 0)

CRIM	<input type="text" value="0.06905"/>	ZN	<input type="text" value="0"/>	INDUS	<input type="text" value="2.18"/>
CHAS	<input type="text" value="0"/>	NOX	<input type="text" value="0.458"/>	RM	<input type="text" value="7.147"/>
AGE	<input type="text" value="54.2"/>	DIS	<input type="text" value="6.0622"/>	RAD	<input type="text" value="3"/>
TAX	<input type="text" value="222"/>	PTRATIO	<input type="text" value="18.7"/>	B	<input type="text" value="396.9"/>
LSTAT	<input type="text" value="5.33"/>				

Run Inference

### Model Predictions

#### Sample Values

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
Original Sample	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.9	5.33
Modified Sample	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.9	5.33

☒ Show all features

#### Model Predictions

	Prediction (True value: 36.2)
Original Sample	32.50857142857136
Modified Sample	32.50857142857136

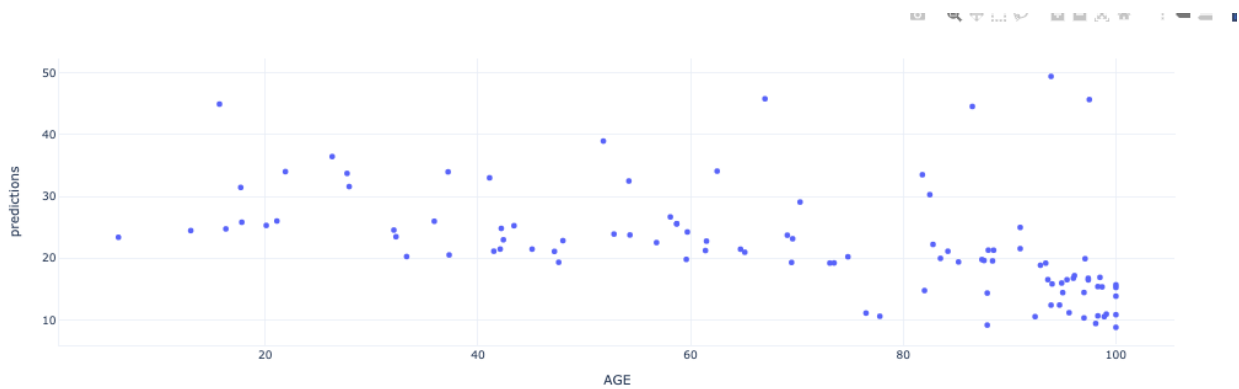
The Predictions Explorer method, `explore_predictions()`, allows the exploration of model predictions across either the marginal distribution (1-feature) or the joint distribution (2-features).

The method `explore_predictions()` has several optional parameters including:

- **discretization:** (str, optional) Discretization method applies the x-axis if the feature **x** is continuous. The valid options are 'quartile', 'decile', or 'percentile'. The default is None.
- **label:** (str or int, optional) Target label or target class name to explore only for classification problems. The default is None.
- **plot\_type:** (str, optional) Type of plot. For classification problems the valid options are 'scatter', 'box', or 'bar'. For a regression problem, the valid options are 'scatter' or 'box'. The default is 'scatter'.
- **x:** (str, optional) Feature column on x-axis. The default is None.
- **y:** (str, optional) Feature column or model prediction column on the y-axis, by default it is the target.

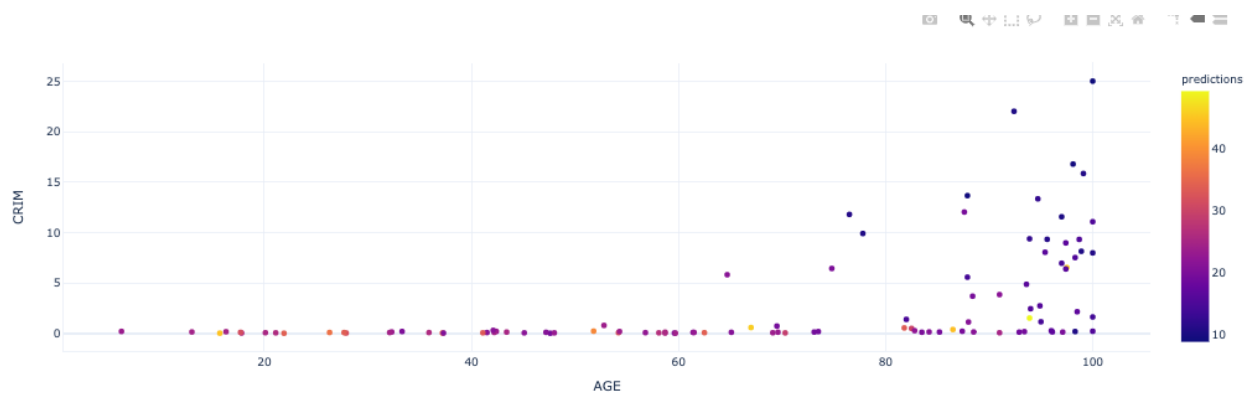
When only **x** is set, the chart shows the relationship between the features **x** and the target **y**.

```
whatif_explainer.explore_predictions(x='AGE')
```

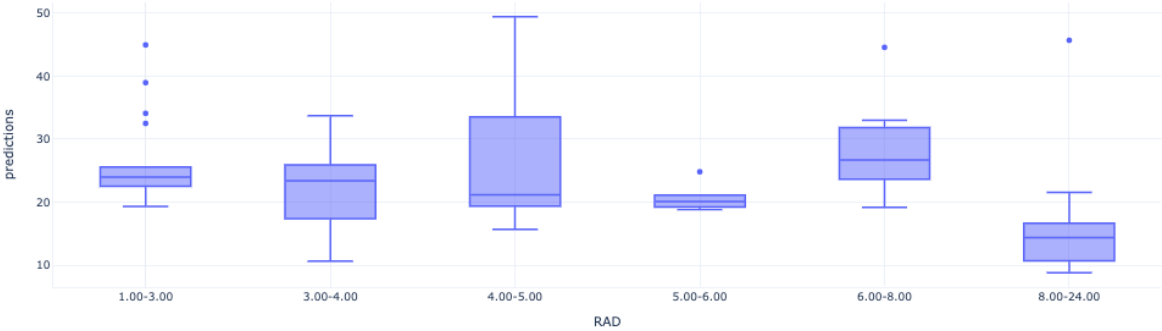


If features are specified for both **x** and **y**, the plot uses color to indicate the value of the target.

```
whatif_explainer.explore_predictions(x='AGE', y='CRIM')
```



```
whatif_explainer.explore_predictions(x='RAD', plot_type='box', discretization='decile')
```





## REGISTER AND DEPLOY MODELS

You could register your model with OCI Data Science service through ADS. Alternatively, the Oracle Cloud Infrastructure (OCI) Console can be used by going to the Data Science projects page, selecting a project, then click **Models**. The models page shows the model artifacts that are in the model catalog for a given project.

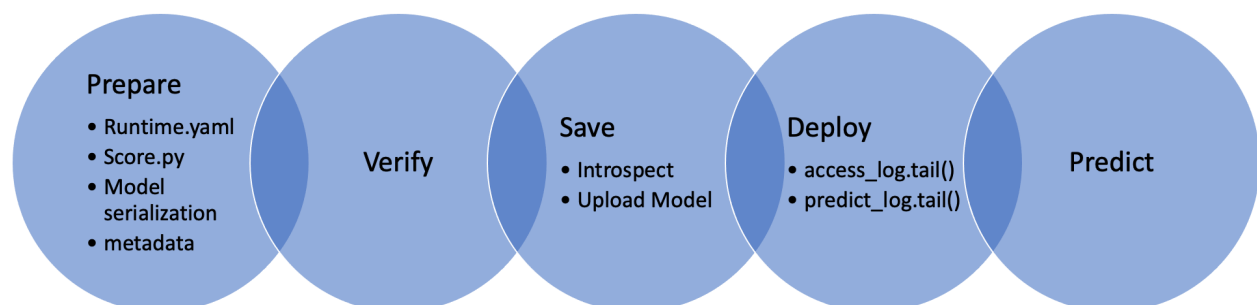
After a model and its artifacts are registered, they become available for other data scientists if they have the correct permissions.

Data scientists can:

- List, read, download, and load models from the catalog to their own notebook sessions.
- Download the model artifact from the catalog, and run the model on their laptop or some other machine.
- Deploy the model artifact as a [model deployment](#).
- Document the model use case and algorithm using taxonomy metadata.
- Add custom metadata that describes the model.
- Document the model provenance including the resources and tags used to create the model (notebook session), and the code used in training.
- Document the input data schema, and the returned inference schema.
- Run introspection tests on the model artifact to ensure that common model artifact errors are flagged. Thus, they can be remediated before the model is saved to the catalog.

The ADS SDK automatically captures some of the metadata for you. It captures provenance, taxonomy, and some custom metadata.

### 12.1 Workflow



ADS has a set of framework specific classes that take your model and push it to production with a few quick steps.

The first step is to create a model serialization object. This object wraps your model and has a number of methods to assist in deploying it. There are different model classes for different model classes. For example, if you have a PyTorch model you would use the `PyTorchModel` class. If you have a TensorFlow model you would use the `TensorFlowModel` class. ADS has model serialization for many different model classes. However, it is not feasible to have a model serialization class for all model types. Therefore, the `GenericModel` can be used for any class that has a `.predict()` method.

After creating the model serialization object, the next step is to use the `.prepare()` method to create the model artifacts. The `score.py` file is created and it is customized to your model class. You may still need to modify it for your specific use case but this is generally not required. The `.prepare()` method also can be used to store metadata about the model, code used to create the model, input and output schema, and much more.

If you make changes to the `score.py` file, call the `.verify()` method to confirm that the `load_model()` and `predict()` functions in this file are working. This speeds up your debugging as you do not need to deploy a model to test it.

The `.save()` method is then used to store the model in the model catalog. A call to the `.deploy()` method creates a load balancer and the instances needed to have an HTTPS access point to perform inference on the model. Using the `.predict()` method, you can send data to the model deployment endpoint and it will return the predictions.

## 12.2 Register

### 12.2.1 Quick Start

ADS can auto generate the required files to register and deploy your models. Checkout the examples below to learn how to deploy models of different frameworks.

#### 12.2.1.1 Sklearn

```
import tempfile
from ads.model.framework.sklearn_model import SklearnModel
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Load dataset and Prepare train and test split
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Train a LogisticRegression model
sklearn_estimator = LogisticRegression()
sklearn_estimator.fit(X_train, y_train)

# Instantiate ads.model.framework.sklearn_model.SklearnModel using the sklearn_
↳LogisticRegression model
sklearn_model = SklearnModel(
    estimator=sklearn_estimator, artifact_dir=tempfile.mkdtemp()
)
```

(continues on next page)



(continued from previous page)

```

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
↪ schema.json
sklearn_model.prepare(
    inference_conda_env="dbexp_p38_cpu_v1",
    X_sample=X_train,
    y_sample=y_train,
)

# Verify generated artifacts
sklearn_model.verify(X_test)

# Register scikit-learn model
model_id = sklearn_model.save(display_name="Sklearn Model")

```

### 12.2.1.2 XGBoost

Create a model, prepare it, verify that it works, save it to the model catalog, deploy it, make a prediction, and then delete the deployment.

```

import tempfile
import xgboost as xgb
from ads.model.framework.xgboost_model import XGBoostModel
from sklearn.datasets import load_iris
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Load dataset and Prepare train and test split
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Train a XBoost Classifier model
xgboost_estimator = xgb.XGBClassifier()
xgboost_estimator.fit(X_train, y_train)

# Instantiate ads.model.framework.xgboost_model.XGBoostModel using the trained XGBoost_
↪ Model
xgboost_model = XGBoostModel(estimator=xgboost_estimator, artifact_dir=tempfile.
↪ mkdtemp())

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
↪ schema.json
xgboost_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    X_sample=X_train,
    y_sample=y_train,
)

# Verify generated artifacts
xgboost_model.verify(X_test)

```

(continues on next page)

(continued from previous page)

```
# Register XGBoost model
model_id = xgboost_model.save(display_name="XGBoost Model")
```

### 12.2.1.3 LightGBM

Create a model, prepare it, verify that it works, save it to the model catalog, deploy it, make a prediction, and then delete the deployment.

```
import lightgbm as lgb
import tempfile
from ads.model.framework.lightgbm_model import LightGBMModel
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset and Prepare train and test split
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Train a XBoost Classifier model
train = lgb.Dataset(X_train, label=y_train)
param = {
    'objective': 'multiclass', 'num_class': 3,
}
lightgbm_estimator = lgb.train(param, train)

# Instantiate ads.model.lightgbm_model.XGBoostModel using the trained LGBM Model
lightgbm_model = LightGBMModel(estimator=lightgbm_estimator, artifact_dir=tempfile.
    ↪mkdtemp())

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
    ↪schema.json
lightgbm_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    X_sample=X_train,
    y_sample=y_train,
)

# Verify generated artifacts
lightgbm_model.verify(X_test)

# Register LightGBM model
model_id = lightgbm_model.save(display_name="LightGBM Model")
```

### 12.2.1.4 PyTorch

Create a model, prepare it, verify that it works, save it to the model catalog, deploy it, make a prediction, and then delete the deployment.

```
import tempfile
import torch
import torchvision
from ads.model.framework.pytorch_model import PyTorchModel

# Load a pre-trained resnet model
torch_estimator = torchvision.models.resnet18(pretrained=True)
torch_estimator.eval()

# create random test data
test_data = torch.randn(1, 3, 224, 224)

# Instantite ads.model.framework.pytorch_model.PyTorchModel using the pre-trained_
↳PyTorch Model
artifact_dir=tempfile.mkdtemp()
torch_model = PyTorchModel(torch_estimator, artifact_dir=artifact_dir)

# Autogenerate score.py, serialized model, runtime.yaml
# Set `use_torch_script` to `True` to save the model as Torchscript program.
torch_model.prepare(inference_conda_env="pytorch110_p38_cpu_v1", use_torch_script=True)

# Verify generated artifacts
torch_model.verify(test_data)

#Register PyTorch model
model_id = torch_model.save(display_name="PyTorch Model")
```

### 12.2.1.5 Spark Pipeline

Create a model, prepare it, verify that it works, save it to the model catalog, deploy it, make a prediction, and then delete the deployment.

```
import tempfile
import os
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from ads.model.framework.spark_model import SparkPipelineModel

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .getOrCreate()

# create data
training = spark.createDataFrame(
```

(continues on next page)

(continued from previous page)

```

[
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0),
],
["id", "text", "label"],
)
test = spark.createDataFrame(
    [
        (4, "spark i j k"),
        (5, "l m n"),
        (6, "spark hadoop spark"),
        (7, "apache hadoop"),
    ],
    ["id", "text"],
)

# Train a Spark Pipeline model
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
model = pipeline.fit(training)

# Instantiate ads.model.framework.spark_model.SparkPipelineModel using the pre-trained
↳ Spark Pipeline Model
spark_model = SparkPipelineModel(estimator=model, artifact_dir=tempfile.mkdtemp())
spark_model.prepare(inference_conda_env="pyspark32_p38_cpu_v2",
                    X_sample = training,
                    force_overwrite=True)

# Verify generated artifacts
prediction = spark_model.verify(test)

# Register Spark model
spark_model.save(display_name="Spark Pipeline Model")

```

### 12.2.1.6 TensorFlow

Create a model, prepare it, verify that it works, save it to the model catalog, deploy it, make a prediction, and then delete the deployment.

```

from ads.model.framework.tensorflow_model import TensorFlowModel
import tempfile
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

```

(continues on next page)

(continued from previous page)

```

tf_estimator = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10),
    ]
)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
tf_estimator.compile(optimizer="adam", loss=loss_fn, metrics=["accuracy"])
tf_estimator.fit(x_train, y_train, epochs=1)

# Instantiate ads.model.framework.tensorflow_model.TensorFlowModel using the pre-trained
↳ TensorFlow Model
tf_model = TensorFlowModel(tf_estimator, artifact_dir=tempfile.mkdtemp())

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
↳ schema.json
tf_model.prepare(inference_conda_env="tensorflow28_p38_cpu_v1")

# Verify generated artifacts
tf_model.verify(x_test[:1])

# Register TensorFlow model
model_id = tf_model.save(display_name="TensorFlow Model")

```

### 12.2.1.7 Other Frameworks

```

import tempfile
from ads.model.generic_model import GenericModel

# Create custom framework model
class Toy:
    def predict(self, x):
        return x ** 2
model = Toy()

# Instantiate ads.model.generic_model.GenericModel using the trained Custom Model
generic_model = GenericModel(estimator=model, artifact_dir=tempfile.mkdtemp())
generic_model.summary_status()

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
↳ schema.json
generic_model.prepare(
    inference_conda_env="dbexp_p38_cpu_v1",
    model_file_name="toy_model.pkl",
    force_overwrite=True
)

# Check if the artifacts are generated correctly.

```

(continues on next page)

(continued from previous page)

```
# The verify method invokes the ``predict`` function defined inside ``score.py`` in the
↪ artifact_dir
generic_model.verify([2])

# Register the model
model_id = generic_model.save(display_name="Custom Framework Model")
```

## 12.2.2 Model Registration

### 12.2.2.1 Model Artifact

To save a trained model on OCI Data Science, prepare a **Model Artifact**.

Model Artifact is a zip file which contains the following artifacts -

- Serialized model or models
- `runtime.yaml` - This yaml captures provenance information and deployment conda environment
- `score.py` - Entry module which is used by the model deployment server to load the model and run prediction
- `input_schema.json` - Describes the schema of the features that will be used within predict function
- `output_schema.json` - Describes the schem of the prediction values
- Any other artifcat that are required during inference time.

ADS can auto generate all the mandatory files to help save the models that are compliant with the OCI Data Science Model Deployment service.

Auto generation of `score.py` with framework specific code for loading models and fetching prediction is available for following frameworks-

- scikit-learn
- XGBoost
- LightGBM
- PyTorch
- SparkPipelineModel
- TensorFlow

To accomodate for other frameworks that are unknown to ADS, a template code for `score.py` is generated in the provided artifcat directory location.

#### 12.2.2.2 Prepare the Model Artifact

To prepare the model artifact -

- Train a model using the framework of your choice
- Create a Model object from one of the framework specific Models available under `ads.model.framework.*`. The Model class takes two parameters - estimator object and a directory location to store autogenerated artifacts.
- call `prepare()` to generate all the files.

See [API documentation](#) for more details about the parameters.

Here is an example for preparing a model artifact for TensorFlow model.

```
from ads.model.framework.tensorflow_model import TensorFlowModel
import tempfile
import tensorflow as tf
from ads.common.model_metadata import UseCaseType

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

tf_estimator = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10),
    ]
)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
tf_estimator.compile(optimizer="adam", loss=loss_fn, metrics=["accuracy"])
tf_estimator.fit(x_train, y_train, epochs=1)

tf_model = TensorFlowModel(tf_estimator, artifact_dir=tempfile.mkdtemp())

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
# → schema.json
tf_model.prepare(inference_conda_env="generalml_p38_cpu_v1",
                 use_case_type=UseCaseType.MULTINOMIAL_CLASSIFICATION,
                 X_sample=trainx,
                 y_sample=trainy
                )

# Verify generated artifacts
tf_model.verify(x_test[:1])

# Register TensorFlow model
model_id = tf_model.save()
```

```
['output_schema.json', 'score.py', 'runtime.yaml', 'model.h5', '.model-ignore', 'input_
→ schema.json']
```

ADS automatically captures:

- Provenance metadata - commit id, git branch, etc
- Taxonomy metadata such as model hyperparameters, framework name.
- Custom metadata such as Data science conda environment when available, Model artifact inventory, model serialization format, etc.
- Schema of input and target variables. This requires input sample and target sample to be passed while calling prepare

**Note:**

- UseCaseType in metadata\_taxonomy cannot be automatically populated. One way to populate the use case is to pass use\_case\_type to the prepare method.
- Model introspection is automatically triggered.

### 12.2.2.3 score.py

In the prepare step, the service automatically generates a `score.py` file in the artifact directory.

`score.py` is used by the Data Science Model Deployment service to generate predictions in the input feature. Here is a minimal `score.py` implementation -

```
import joblib
model_name = "model.joblib"

def load_model(): # load_model must mandatorily return ``not None`` object.

    model = None
    with open(os.path.join(os.path.dirname(os.path.realpath(__file__)), model_file_name),
        ↪ "rb") as mfile:
        model = joblib.load(mfile)
    return model

def predict(data, model=load_model()):
    return model.predict(data).tolist()
```

ADS autogenerates framework specific `score.py` which provides following functionality -

- Parse the input data and convert to pandas dataframe/numpy array/list
- Ensure the data type after converting to pandas dataframe matches the training time. This is achieved using the schema definition generated during prepare step.
- Serialize prediction generated by the model such that it is json serializable to avoid deployment runtime errors

You could customize the `score.py` to fit your use case. The most common use case for changing the `score.py` file is to add preprocessing and postprocessing steps to the `predict()` method.

Refer Cusotmization section for how to change and verify the model artifacts.

The `score.py` consists of multiple functions among which the `load_model` and `predict` are most important.

#### 12.2.2.3.1 load\_model

During deployment, the `load_model` method loads the serialized model. The `load_model` method is always fully populated, except when you set `serialize=False` for `GenericModel`.

- For the `GenericModel` class, if you choose `serialize=True` in the init function, the model is pickled and the `score.py` is fully auto-populated to support loading the pickled model. Otherwise, the user is responsible to fill the `load_model`.
- For other frameworks, this part is fully populated.

**Note:** `load_model` should return not `None` value for successful deployment.



### 12.2.2.3.2 predict

The `predict` method is triggered every time a payload is sent to the model deployment endpoint. The method takes the payload and the loaded model as inputs. Based on the payload, the method returns the predicted results output by the model.

### 12.2.2.3.3 pre\_inference

If the payload passed to the endpoint needs preprocessing, this function does the preprocessing step. The user is fully responsible for the preprocessing step.

### 12.2.2.3.4 post\_inference

If the predicted result from the model needs some postprocessing, the user can put the logic in this function.

### 12.2.2.3.5 deserialize

When you use the `.verify()` or `.predict()` methods from model classes such as `GenericModel` or `SklearnModel`, if the data passed in is not in bytes or `JsonSerializable`, the models try to serialize the data. For example, if a pandas dataframe is passed and not accepted by the deployment endpoint, the pandas dataframe is converted to JSON internally. When the `X_sample` variable is passed into the `.prepare()` function, the data type of pandas dataframe is passed to the endpoint, and the schema of the dataframe is recorded in the `input_schema.json` file. Then, the JSON payload is sent to the endpoint. Because the model expects to take a pandas dataframe, the `.deserialize()` method converts the JSON back to the pandas dataframe using the schema and the data type. For all frameworks except for the `GenericModel` class, the `.deserialize()` method is auto-populated. Note that for each framework, only specific data types are supported.

Starting from `.. versionadded:: 2.6.3`, you can send the bytes to the endpoint directly. If the bytes payload is sent to the endpoint, bytes are passed directly to the model. If the model expects a specific data format, you need to write the conversion logic yourself.

### 12.2.2.3.6 fetch\_data\_type\_from\_schema

This function is used to load the schema from the `input_schema.json` when needed.

## 12.2.2.4 Model Introspection

The `.introspect()` method runs some sanity checks on the `runtime.yaml`, and `score.py` files. This is to help you identify potential errors that might occur during model deployment. It checks fields such as environment path, validates the path's existence on the Object Storage, checks if the `.load_model()`, and `.predict()` functions are defined in `score.py`, and so on. The result of model introspection is automatically saved to the taxonomy metadata and model artifacts.

```
tf_model.introspect()
```

```
['output_schema.json', 'runtime.yaml', 'model.joblib', 'input_schema.json', 'score.py']
```

	Test key	Test name	Result	Message
0	runtime_env_path	Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_PATH is set	Passed	
1	runtime_env_python	Check that field MODEL_DEPLOYMENT.INFERENCE_PYTHON_VERSION is set to a value of 3.6 or higher	Passed	
2	runtime_env_slug	Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG is set	Passed	
3	runtime_env_type	Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is set to a value in (published, data_science)	Passed	
4	runtime_path_exist	If MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is data_science and MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG is set, check that the file path in MODEL_DEPLOYMENT.INFERENCE_ENV_PATH is correct.	Skipped	
5	runtime_slug_exist	If MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is data_science, check that the slug listed in MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG exists.	Skipped	
6	runtime_version	Check that field MODEL_ARTIFACT_VERSION is set to 3.0	Passed	
7	runtime_yaml	Check that the file "runtime.yaml" exists and is in the top level directory of the artifact directory	Passed	
8	score_load_model	Check that load_model() is defined	Passed	
9	score_predict	Check that predict() is defined	Passed	
10	score_predict_arg	Check that all other arguments in predict() are optional and have default values	Passed	
11	score_predict_data	Check that the only required argument for predict() is named "data"	Passed	
12	score_py	Check that the file "score.py" exists and is in the top level directory of the artifact directory	Passed	
13	score_syntax	Check for Python syntax errors	Passed	

Reloading model artifacts automatically invokes model introspection. However, you can invoke introspection manually by calling `tf_model.introspect()`:

The `ArtifactTestResults` field is populated in `metadata_taxonomy` when `instrospect` is triggered:

```
tf_model.metadata_taxonomy['ArtifactTestResults']
```

```
key: ArtifactTestResults
value:
  runtime_env_path:
    category: conda_env
    description: Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_PATH is set
  ...
```

### 12.2.2.5 Save Model

The `.save()` method saves the model artifacts, introspection results, schema, metadata, etc on OCI Data Science Service and returns a model OCID. See [API documentation](#) for more details.

```
model_catalog_id = tf_model.save(display_name='TF Model')
```

You can print a model to see some details about it.

```
print(tf_model)
```

You can also print a summary status of the model.

```
lgbm_model.summary_status()
```

### 12.2.2.6 Save Large Model

New in version 2.6.4.

Large models are models with artifacts between 2 and 6 GB. You must first upload large models to an Object Storage bucket, and then transfer them to a model catalog.

ADS *framework specific wrapper* classes save large models using a process almost identical to model artifacts that are less than 2GB. An Object Storage bucket is required with Data Science service access granted to that bucket.

If you don't have an Object Storage bucket, create one using the OCI SDK or the Console. Create an [Object Storage bucket](#). Make a note of the namespace, compartment, and bucket name. Configure the following policies to allow the Data Science service to read and write the model artifact to the Object Storage bucket in your tenancy. An administrator must configure these policies in [IAM](#) in the Console.

```
Allow service datascience to manage object-family in compartment <compartment> where ALL
↪ {target.bucket.name='<bucket_name>'}
```

```
Allow service objectstorage to manage object-family in compartment <compartment> where
↪ ALL {target.bucket.name='<bucket_name>'}
```

See [API documentation](#) for more details.

The following saves the *framework specific wrapper* object, `model`, to the model catalog and returns the OCID from the model catalog:

```
model_catalog_id = tf_model.save(
    display_name='TF Model'
    bucket_uri=<oci://<bucket_name>@<namespace>/<path>/>,
    overwrite_existing_artifact = True,
    remove_existing_artifact = True,
)
```

### 12.2.2.7 Export Model Artifact to Object Storage

The model artifacts can be uploaded to the Object Storage bucket. The `.upload_artifact()` method archives all files located in the artifact folder and uploads it to the Object Storage bucket.

```
tf_model.upload_artifact(uri="<oci://bucket@namespace/prefix/tf_model.zip>")
```

Uploaded artifacts can be used to *load* and recreate *framework specific wrapper* objects from the existing model artifact archive. To construct the model back from the existing artifact, use the `.from_model_artifact()` method.

```
from ads.model.framework.tensorflow_model import TensorFlowModel

tf_model = TensorFlowModel.from_model_artifact(
    "<oci://bucket@namespace/prefix/tf_model.zip>"
)
```

### 12.2.3 Model Schema

The data schema provides a definition of the format and nature of the data that the model expects. It also defines the output data from the model inference. The `.populate_schema()` method accepts the parameters, `data_sample` or `X_sample`, and `y_sample`. When using these parameters, the model artifact gets populated the input and output data schemas.

The `.schema_input` and `.schema_output` properties are Schema objects that define the schema of each input column and the output. The Schema object contains these fields:

- **description:** Description of the data in the column.
- **domain:** A data structure that defines the domain of the data. The restrictions on the data and summary statistics of its distribution.
  - **constraints:** A data structure that is a list of expression objects that defines the constraints of the data.
    - \* **expression:** A string representation of an expression that can be evaluated by the language corresponding to the value provided in `language` attribute. The default value for `language` is `python`.
      - **expression:** Required. Use the `string.Template` format for specifying the expression. `$x` is used to represent the variable.
      - **language:** The default value is `python`. Only `python` is supported.
  - **stats:** A set of summary statistics that defines the distribution of the data. These are determined using the feature type statistics as defined in ADS.
  - **values:** A description of the values of the data.
- **dtype:** Pandas data type
- **feature\_type:** The primary feature type as defined by ADS.
- **name:** Name of the column.
- **required:** Boolean value indicating if a value is always required.

```
- description: Number of matching socks in your dresser drawer.
domain:
  constraints:
  - expression: ($x <= 10) and ($x > 0)
    language: python
  - expression: $x in [2, 4, 6, 8, 10]
    language: python
  stats:
    count: 465.0
    lower_quartile: 3.2
    mean: 6.3
    median: 7.0
    sample_maximum: 10.0
    sample_minimum: 2.0
    standard_deviation: 2.5
    upper_quartile: 8.2
  values: Natural even numbers that are less than or equal to 10.
dtype: int64
feature_type: EvenNatural10
name: sock_count
required: true
```

## 12.2.3.1 Schema Model

```

{
  "description": {
    "nullable": true,
    "required": false,
    "type": "string"
  },
  "domain": {
    "nullable": true,
    "required": false,
    "schema": {
      "constraints": {
        "nullable": true,
        "required": false,
        "type": "list"
      },
      "stats": {
        "nullable": true,
        "required": false,
        "type": "dict"
      },
      "values": {
        "nullable": true,
        "required": false,
        "type": "string"
      }
    },
    "type": "dict"
  },
  "dtype": {
    "nullable": false,
    "required": true,
    "type": "string"
  },
  "feature_type": {
    "nullable": true,
    "required": false,
    "type": "string"
  },
  "name": {
    "nullable": false,
    "required": true,
    "type": [
      "string",
      "number"
    ]
  },
  "order": {
    "nullable": true,
    "required": false,
    "type": "integer"
  },
}

```

(continues on next page)

(continued from previous page)

```

"required": {
  "nullable": false,
  "required": true,
  "type": "boolean"
}
}

```

### 12.2.3.2 Generating Schema

To auto generate schema from the training data, provide X sample and the y sample while preparing the model artifact.

Eg.

```

import tempfile
from ads.model.framework.sklearn_model import SklearnModel
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Load dataset and Prepare train and test split
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Train a LogisticRegression model
sklearn_estimator = LogisticRegression()
sklearn_estimator.fit(X_train, y_train)

# Instantite ads.model.SklearnModel using the sklearn LogisticRegression model
sklearn_model = SklearnModel(estimator=sklearn_estimator, artifact_dir=tempfile.
↳mkdtemp())

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
↳schema.json
sklearn_model.prepare(inference_conda_env="dataexpl_p37_cpu_v3", X_sample=trainx, y_
↳sample=trainy)

```

Calling `.schema_input` or `.schema_output` shows the schema in a YAML format.

Alternatively, you can check the `output_schema.json` file for the content of the `schema_output`:

```

with open(path.join(path_to_artifact_dir, "output_schema.json"), 'r') as f:
    print(f.read())

```

```

{
  "schema": [
    {
      "dtype": "int64",
      "feature_type": "Integer",
      "name": "class",
      "domain": {
        "values": "Integer",

```

(continues on next page)

(continued from previous page)

```

        "stats": {
            "count": 465.0,
            "mean": 0.5225806451612903,
            "standard deviation": 0.5000278079030275,
            "sample minimum": 0.0,
            "lower quartile": 0.0,
            "median": 1.0,
            "upper quartile": 1.0,
            "sample maximum": 1.0
        },
        "constraints": [],
    },
    "required": true,
    "description": "class"
}
]
}

```

### 12.2.3.3 Update the Schema

You can update the fields in the schema:

```

sklearn_model.schema_output[<class name>].description = 'target variable'
sklearn_model.schema_output[<class name>].feature_type = 'Category'

```

You can specify a constraint for your data using Expression, and call evaluate to check if the data satisfies the constraint:

```

sklearn_model.schema_input['col01'].domain.constraints.append(Expression('($x < 20) and (
↪ $x > -20)'))

```

0 is between -20 and 20, so evaluate should return True:

```

sklearn_model.schema_input['col01'].domain.constraints[0].evaluate(x=0)

```

```

True

```

You can directly populate the schema by calling populate\_schema():

```

sklearn_model.model_artifact.populate_schema(X_sample=test.X, y_sample=test.y)

```

You can also load your schema from a JSON or YAML file:

```

cat <<EOF > schema.json
{
    "schema": [
        {
            "dtype": "int64",
            "feature_type": "Category",
            "name": "class",
            "domain": {

```

(continues on next page)

(continued from previous page)

```

        "values": "Category type.",
        "stats": {
            "count": 465.0,
            "unique": 2},
        "constraints": [
            {"expression": "($x <= 1) and ($x >= 0)", "language": "python"},
            {"expression": "$x in [0, 1]", "language": "python"}]},
        "required": true,
        "description": "target to predict."
    }
]
}
EOF

```

```
sklearn_model.schema_output = Schema.from_file('schema.json'))
```

## 12.2.4 Model Metadata

When you register a model, you can add metadata to help with the documentation of the model. Service defined metadata fields are known as **Taxonomy Metadata** and user defined metadata fields are known as **Custom Metadata**

### 12.2.4.1 Taxonomy Metadata

Taxonomy metadata includes the type of the model, use case type, libraries, framework, and so on. This metadata provides a way of documenting the schema of the model. The `UseCaseType`, `FrameWork`, `FrameWorkVersion`, `Algorithm`, and `Hyperparameters` are fixed taxonomy metadata. These fields are automatically populated when the `.prepare()` method is called. You can also manually update the values of those fields.

- `ads.common.model_metadata.UseCaseType`: The machine learning problem associated with the Estimator class. The `UseCaseType.values()` method returns the most current list. This is a list of allowed values.:
  - `UseCaseType.ANOMALY_DETECTION`
  - `UseCaseType.BINARY_CLASSIFICATION`
  - `UseCaseType.CLUSTERING`
  - `UseCaseType.DIMENSIONALITY_REDUCTION`
  - `UseCaseType.IMAGE_CLASSIFICATION`
  - `UseCaseType.MULTINOMIAL_CLASSIFICATION`
  - `UseCaseType.NER`
  - `UseCaseType.OBJECT_LOCALIZATION`
  - `UseCaseType.OTHER`
  - `UseCaseType.RECOMMENDER`
  - `UseCaseType.REGRESSION`
  - `UseCaseType.SENTIMENT_ANALYSIS`
  - `UseCaseType.TIME_SERIES_FORECASTING`
  - `UseCaseType.TOPIC_MODELING`



- `ads.common.model_metadata.FrameWork`: The `FrameWork` of the estimator object. You can get the list of allowed values using `Framework.values()`:
  - `Framework.BERT`
  - `Framework.CUML`
  - `Framework.EMCEE`
  - `Framework.ENSEMBLE`
  - `Framework.FLAIR`
  - `Framework.GENSIM`
  - `Framework.H2O`
  - `Framework.KERAS`
  - `Framework.LIGHTgbm`
  - `Framework.MXNET`
  - `Framework.NLTK`
  - `Framework.ORACLE_AUTOML`
  - `Framework.OTHER`
  - `Framework.PROPHET`
  - `Framework.PYOD`
  - `Framework.PYMC3`
  - `Framework.PYSTAN`
  - `Framework.PYTORCH`
  - `Framework.SCIKIT_LEARN`
  - `Framework.SKTIME`
  - `Framework.SPACY`
  - `Framework.STATSMODELS`
  - `Framework.TENSORFLOW`
  - `Framework.TRANSFORMERS`
  - `Framework.WORD2VEC`
  - `Framework.XGBOOST`
- `FrameworkVersion`: The framework version of the estimator object. For example, 2.3.1.
- `Algorithm`: The model class.
- `Hyperparameters`: The hyperparameters of the estimator object.

You can't add or delete any of the fields, or change the key of those fields.

You can populate the `use_case_type` by passing it in the `.prepare()` method. Or you can set and update it directly.

```
import tempfile
from ads.model.framework.sklearn_model import SklearnModel
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
```

(continues on next page)

(continued from previous page)

```

from sklearn.model_selection import train_test_split
from ads.common.model_metadata import UseCaseType

# Load dataset and Prepare train and test split
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Train a LogisticRegression model
sklearn_estimator = LogisticRegression()
sklearn_estimator.fit(X_train, y_train)

# Instantiate ads.model.SklearnModel using the sklearn LogisticRegression model
sklearn_model = SklearnModel(estimator=sklearn_estimator, artifact_dir=tempfile.
    ↳mkdtemp())

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
    ↳schema.json
sklearn_model.prepare(inference_conda_env="dataexpl_p37_cpu_v3", X_sample=trainx, y_
    ↳sample=trainy)

sklearn_model.metadata_taxonomy['UseCaseType'].value = UseCaseType.BINARY_CLASSIFICATION

```

### Update metadata\_taxonomy

Update any of the taxonomy fields with allowed values:

```

sklearn_model.metadata_taxonomy['FrameworkVersion'].value = '0.24.2'
sklearn_model.metadata_taxonomy['UseCaseType'].update(value=UseCaseType.BINARY_
    ↳CLASSIFICATION)

```

You can view the metadata\_taxonomy in the dataframe format by calling to\_dataframe:

```
sklearn_model.metadata_taxonomy.to_dataframe()
```

	Key	Value
0	Algorithm	RandomForestClassifier
1	ArtifactTestResults	{'score_py': {'key': 'score_py', 'category': 'Mandatory Files Check', 'description': 'Check that the file "score.py" exists and is in the top level directory of the artifact directory', 'error_msg': 'The file "score.py" is missing.', 'success': True}, 'runtime_yaml': {'category': 'Mandatory Files Check', 'description': 'Check that the file "runtime.yaml" exists and is in the top level directory of the artifact directory', 'error_msg': 'The file "runtime.yaml" is missing.', 'success': True}, 'score_syntax': {'category': 'score.py', 'description': 'Check for Python syntax errors', 'error_msg': 'There is Syntax error in score.py.', 'success': True}, 'score_load_model': {'category': 'score.py', 'description': 'Check that load_model() is defined', 'error_msg': 'Function load_model is not present in score.py.', 'success': True}, 'score_predict': {'category': 'score.py', 'description': 'Check that predict() is defined', 'error_msg': 'Function predict is not present in score.py.', 'success': True}, 'score_predict_data': {'category': 'score.py', 'description': 'Check that the only required argument for predict() is named "data"', 'error_msg': 'The predict function in score.py must have a formal argument named "data"', 'success': True}, 'score_predict_arg': {'category': 'score.py', 'description': 'Check that all other arguments in predict() are optional and have default values', 'error_msg': 'All formal arguments in the predict function must have default values, except that "data" argument.', 'success': True}, 'runtime_version': {'category': 'runtime.yaml', 'description': 'Check that field MODEL_ARTIFACT_VERSION is set to 3.0', 'error_msg': 'In runtime.yaml, the key MODEL_ARTIFACT_VERSION must be set to 3.0.', 'success': True}, 'runtime_env_python': {'category': 'conda_env', 'description': 'Check that field MODEL_DEPLOYMENT.INFERENCE_PYTHON_VERSION is set to a value of 3.6 or higher', 'error_msg': 'In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_PYTHON_VERSION must be set to a value of 3.6 or higher.', 'success': True, 'value': '3.7.10'}, 'runtime_env_type': {'category': 'conda_env', 'description': 'Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is set to a value in (published, data_science)', 'error_msg': 'In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE must be set to published or data_science.', 'success': True, 'value': 'published'}, 'runtime_env_slug': {'category': 'conda_env', 'description': 'Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG is set', 'error_msg': 'In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG must have a value.', 'success': True, 'value': 'database_p37_cpu_v1.0'}, 'runtime_env_path': {'category': 'conda_env', 'description': 'Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_PATH is set', 'error_msg': 'In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_ENV_PATH must have a value.', 'success': True, 'value': 'oci://licence_checker@ociodscdev/conda_environments/cpu/OracleDatabase/1.0/database_p37_cpu_v1.0'}, 'runtime_path_exists': {'category': 'conda_env', 'description': 'If MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is data_science and MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG is set, check that the file path in MODEL_DEPLOYMENT.INFERENCE_ENV_PATH is correct.', 'error_msg': 'In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_ENV_PATH does not exist.', 'runtime_slug_exists': {'category': 'conda_env', 'description': 'If MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is data_science, check that the slug listed in MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG exists.', 'error_msg': 'In runtime.yaml, the value of the key INFERENC...'}, 'slug_value' and it doesn't exist in the bucket 'bucket_url'. Ensure that the value INFERENC...}}}
2	Framework	scikit-learn
3	FrameworkVersion	0.24.2
4	Hyperparameters	{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}
5	UseCaseType	binary_classification

Alternatively, you can view it directly in a YAML format:

```
sklearn_model.metadata_taxonomy
```

```
data:
- key: FrameworkVersion
  value: 0.24.2
- key: ArtifactTestResults
  value:
    runtime_env_path:
      category: conda_env
      description: Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_PATH is set
      error_msg: In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_ENV_PATH must
        have a value.
      success: true
      value: oci://licence_checker@ociodscdev/conda_environments/cpu/Oracle Database/1.0/
      ↪ database_p37_cpu_v1.0
    runtime_env_python:
      category: conda_env
      description: Check that field MODEL_DEPLOYMENT.INFERENCE_PYTHON_VERSION is set
        to a value of 3.6 or higher
      error_msg: In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_PYTHON_VERSION
        must be set to a value of 3.6 or higher.
      success: true
      value: 3.7.10
    runtime_env_slug:
      category: conda_env
      description: Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG is set
      error_msg: In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG must
        have a value.
      success: true
      value: database_p37_cpu_v1.0
    runtime_env_type:
      category: conda_env
      description: Check that field MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is set to
        a value in (published, data_science)
      error_msg: In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE must
        be set to published or data_science.
      success: true
      value: published
    runtime_path_exist:
      category: conda_env
      description: If MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is data_science and MODEL_
      ↪ DEPLOYMENT.INFERENCE_ENV_SLUG
        is set, check that the file path in MODEL_DEPLOYMENT.INFERENCE_ENV_PATH is
        correct.
      error_msg: In runtime.yaml, the key MODEL_DEPLOYMENT.INFERENCE_ENV_PATH does
        not exist.
    runtime_slug_exist:
      category: conda_env
      description: If MODEL_DEPLOYMENT.INFERENCE_ENV_TYPE is data_science, check that
        the slug listed in MODEL_DEPLOYMENT.INFERENCE_ENV_SLUG exists.
      error_msg: In runtime.yaml, the value of the key INFERENCE_ENV_SLUG is slug_value
        and it doesn't exist in the bucket bucket_url. Ensure that the value INFERENCE_
```

```
→ENV_SLUG
    and the bucket url are correct.
runtime_version:
  category: runtime.yaml
  description: Check that field MODEL_ARTIFACT_VERSION is set to 3.0
  error_msg: In runtime.yaml, the key MODEL_ARTIFACT_VERSION must be set to 3.0.
  success: true
runtime_yaml:
  category: Mandatory Files Check
  description: Check that the file "runtime.yaml" exists and is in the top level
    directory of the artifact directory
  error_msg: The file 'runtime.yaml' is missing.
  success: true
score_load_model:
  category: score.py
  description: Check that load_model() is defined
  error_msg: Function load_model is not present in score.py.
  success: true
score_predict:
  category: score.py
  description: Check that predict() is defined
  error_msg: Function predict is not present in score.py.
  success: true
score_predict_arg:
  category: score.py
  description: Check that all other arguments in predict() are optional and have
    default values
  error_msg: All formal arguments in the predict function must have default values,
    except that 'data' argument.
  success: true
score_predict_data:
  category: score.py
  description: Check that the only required argument for predict() is named "data"
  error_msg: The predict function in score.py must have a formal argument named
    'data'.
  success: true
score_py:
  category: Mandatory Files Check
  description: Check that the file "score.py" exists and is in the top level.
→directory
    of the artifact directory
    error_msg: The file 'score.py' is missing.
    key: score_py
    success: true
score_syntax:
  category: score.py
  description: Check for Python syntax errors
  error_msg: 'There is Syntax error in score.py: '
  success: true
- key: Framework
  value: scikit-learn
- key: UseCaseType
  value: binary_classification
- key: Algorithm
```

```

value: RandomForestClassifier
- key: Hyperparameters
value:
  bootstrap: true
  ccp_alpha: 0.0
  class_weight: null
  criterion: gini
  max_depth: null
  max_features: auto
  max_leaf_nodes: null
  max_samples: null
  min_impurity_decrease: 0.0
  min_impurity_split: null
  min_samples_leaf: 1
  min_samples_split: 2
  min_weight_fraction_leaf: 0.0
  n_estimators: 10
  n_jobs: null
  oob_score: false
  random_state: null
  verbose: 0
  warm_start: false

```

#### 12.2.4.2 Custom Metadata

Update your custom metadata using the key, value, category, and description fields. The key, and value fields are required.

You can see the allowed values for custom metadata category using `MetadataCustomCategory.values()`:

- `MetadataCustomCategory.PERFORMANCE`
- `MetadataCustomCategory.TRAINING_PROFILE`
- `MetadataCustomCategory.TRAINING_AND_VALIDATION_DATASETS`
- `MetadataCustomCategory.TRAINING_ENVIRONMENT`
- `MetadataCustomCategory.OTHER`

#### Add New Custom Metadata

To add a new custom metadata, call `.add()`:

```

sklearn_model.metadata_custom.add(key='test', value='test',
↪category=MetadataCustomCategory.OTHER, description='test', replace=True)

```

#### Update Custom Metadata

Use the `.update()` method to update the fields of a specific key ensuring that you pass all the values you need in the update:

```

sklearn_model.metadata_custom['test'].update(value='test1', description=None,
↪category=MetadataCustomCategory.TRAINING_ENV)

```

Alternatively, you can set it directly:

```
sklearn_model.metadata_custom['test'].value = 'test1'
sklearn_model.metadata_custom['test'].description = None
sklearn_model.metadata_custom['test'].category = MetadataCustomCategory.TRAINING_ENV
```

You can view the custom metadata in the dataframe by calling `.to_dataframe()`:

```
sklearn_model.metadata_custom.to_dataframe()
```

	Key	Value	Description	Category
0	ClientLibrary	ADS		Other
1	CondaEnvironment	database_p37_cpu_v1.0	The conda env where model was trained	Training Environment
2	CondaEnvironmentPath	oci://licence_checker@ociodscdev/conda_environments/cpu/Oracle Database/1.0/database_p37_cpu_v1.0	The oci path of the conda env where model was trained	Training Environment
3	EnvironmentType	published	The env type, could be published conda or datascience conda	Training Environment
4	ModelArtifacts	score.py, runtime.yaml, onnx_data_transformer.json, model.onnx, .model-ignore	The list of files located in artifacts folder	Training Environment
5	ModelSerializationFormat	onnx	The model serialization format	Training Profile
6	SlugName	database_p37_cpu_v1.0	The slug name of the conda env where model was trained	Training Environment
7	test	test1	None	Training Environment

Alternatively, you can view the custom metadata in YAML format by calling `.metadata_custom`:

```
sklearn_model.metadata_custom
```

```
data:
- category: Training Environment
  description: The conda env where model was trained
  key: CondaEnvironment
  value: database_p37_cpu_v1.0
- category: Training Environment
  description: null
  key: test
  value: test1
- category: Training Environment
  description: The env type, could be published conda or datascience conda
  key: EnvironmentType
  value: published
- category: Training Environment
  description: The list of files located in artifacts folder
  key: ModelArtifacts
  value: score.py, runtime.yaml, onnx_data_transformer.json, model.onnx, .model-ignore
- category: Training Environment
  description: The slug name of the conda env where model was trained
  key: SlugName
  value: database_p37_cpu_v1.0
- category: Training Environment
  description: The oci path of the conda env where model was trained
  key: CondaEnvironmentPath
  value: oci://licence_checker@ociodscdev/conda_environments/cpu/Oracle Database/1.0/
  ↪ database_p37_cpu_v1.0
- category: Other
  description: ''
  key: ClientLibrary
```

(continues on next page)

(continued from previous page)

```

value: ADS
- category: Training Profile
  description: The model serialization format
  key: ModelSerializationFormat
  value: onnx

```

When the combined total size of `metadata_custom` and `metadata_taxonomy` exceeds 32000 bytes, an error occurs when you save the model to the model catalog. You can save the `metadata_custom` and `metadata_taxonomy` to the artifacts folder:

```
sklearn_model.metadata_custom.to_json_file(path_to_ADS_model_artifact)
```

You can also save individual items from the custom and taxonomy metadata:

```
sklearn_model.metadata_taxonomy['Hyperparameters'].to_json_file(path_to_ADS_model_
↪artifact)
```

If you already have the training or validation dataset saved in Object Storage and want to document this information in this model artifact object, you can add that information into `metadata_custom`:

```

sklearn_model.metadata_custom.set_training_data(path='oci://bucket_name@namespace/train_
↪data_filename', data_size='(200,100)')
sklearn_model.metadata_custom.set_validation_data(path='oci://bucket_name@namespace/
↪validation_data_filename', data_size='(100,100)')

```

## 12.2.5 Customizing the Model

### 12.2.5.1 Customize score.py

Here is an example for preparing a model artifact for TensorFlow model which is trained on the minsit dataset. The final layer in the model produces 10 values corresponding to each digit. The default `score.py` will produce an array of 10 elements for each input vector. Suppose you want to change the default behavior of `predict` function in `score.py` to return most likely digit instead of returning a probability distribution over all the digits. To do so we can return the position corresponding to the maximum value within the output array. Here are the steps to customize the `score.py` -

Step1: Train your estimator and then generate the Model artifact as shown below -

```

from ads.catalog.model import ModelCatalog
from ads.model.framework.tensorflow_model import TensorFlowModel
import tempfile
import tensorflow as tf
from ads.common.model_metadata import UseCaseType

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

tf_estimator = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dropout(0.2),

```

(continues on next page)

(continued from previous page)

```

        tf.keras.layers.Dense(10),
    ]
)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
tf_estimator.compile(optimizer="adam", loss=loss_fn, metrics=["accuracy"])
tf_estimator.fit(x_train, y_train, epochs=1)

tf_model = TensorFlowModel(tf_estimator, artifact_dir=tempfile.mkdtemp())

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
↪ schema.json
tf_model.prepare(inference_conda_env="generalml_p38_cpu_v1",
                 use_case_type=UseCaseType.MULTINOMIAL_CLASSIFICATION,
                 X_sample=trainx,
                 y_sample=trainy
                )

```

Verify the output produced by the autogenerated `score.py` by calling `verify` on `Model` object.

```
print(tensorflow_model.verify(testx[:3])['prediction'])
```

```

[[-2.9461750984191895, -5.293642997741699, 0.4030594229698181, 3.0270071029663086, -6.
↪ 470805644989014, -2.07453989982605, -9.646402359008789, 9.256569862365723, -2.
↪ 6433541774749756, -0.8167083263397217],
[-3.4297854900360107, 2.4863781929016113, 8.968724250793457, 3.162344217300415, -11.
↪ 153030395507812, 0.15335027873516083, -0.5451826453208923, -7.817524433135986, -1.
↪ 0585914850234985, -10.736929893493652],
[-4.420501232147217, 5.841022491455078, -0.17066864669322968, -1.0071465969085693, -2.
↪ 261953592300415, -3.0983355045318604, -2.0874621868133545, 1.0745809078216553, -1.
↪ 2511857748031616, -2.273810625076294]]

```

The default `score.py` in `tf_model.artifact_dir` location is -

```

import os
import sys
from functools import lru_cache
import pandas as pd
import numpy as np
import json
import tensorflow as tf
from io import BytesIO
import base64

model_name = 'model.h5'

"""
Inference script. This script is used for prediction by scoring server when schema is
↪ known.
"""

```

(continues on next page)



(continued from previous page)

```

@lru_cache(maxsize=10)
def load_model(model_file_name=model_name):
    """
    Loads model from the serialized format

    Returns
    -----
    model: a model instance on which predict API can be invoked
    """
    model_dir = os.path.dirname(os.path.realpath(__file__))
    if model_dir not in sys.path:
        sys.path.insert(0, model_dir)
    contents = os.listdir(model_dir)
    if model_file_name in contents:
        print(f'Start loading {model_file_name} from model directory {model_dir} ...')
        loaded_model = tf.keras.models.load_model(os.path.join(model_dir, model_file_
↪name))

        print("Model is successfully loaded.")
        return loaded_model
    else:
        raise Exception(f'{model_file_name} is not found in model directory {model_dir}')

@lru_cache(maxsize=1)
def fetch_data_type_from_schema(input_schema_path=os.path.join(os.path.dirname(os.path.
↪realpath(__file__)), "input_schema.json")):
    """
    Returns data type information fetch from input_schema.json.

    Parameters
    -----
    input_schema_path: path of input schema.

    Returns
    -----
    data_type: data type fetch from input_schema.json.

    """
    data_type = {}
    if os.path.exists(input_schema_path):
        schema = json.load(open(input_schema_path))
        for col in schema['schema']:
            data_type[col['name']] = col['dtype']
    else:
        print("input_schema has to be passed in in order to recover the same data type.
↪pass `X_sample` in `ads.model.framework.tensorflow_model.TensorFlowModel.prepare`
↪function to generate the input_schema. Otherwise, the data type might be changed after
↪serialization/deserialization.")
        return data_type

```

(continues on next page)

(continued from previous page)

```

def deserialize(data, input_schema_path):
    """
    Deserialize json-serialized data to data in original type when sent to
    predict.

    Parameters
    -----
    data: serialized input data.
    input_schema_path: path of input schema.

    Returns
    -----
    data: deserialized input data.

    """
    data_type = data.get('data_type', '')
    json_data = data.get('data', data)

    if "numpy.ndarray" in data_type:
        load_bytes = BytesIO(base64.b64decode(json_data.encode('utf-8')))
        return np.load(load_bytes, allow_pickle=True)
    if "pandas.core.series.Series" in data_type:
        return pd.Series(json_data)
    if "pandas.core.frame.DataFrame" in data_type:
        return pd.read_json(json_data, dtype=fetch_data_type_from_schema(input_schema_
↪path))
    if "tensorflow.python.framework.ops.EagerTensor" in data_type:
        load_bytes = BytesIO(base64.b64decode(json_data.encode('utf-8')))
        return tf.convert_to_tensor(np.load(load_bytes, allow_pickle=True))

    return json_data

def pre_inference(data, input_schema_path):
    """
    Preprocess json-serialized data to feed into predict function.

    Parameters
    -----
    data: Data format as expected by the predict API of the core estimator.
    input_schema_path: path of input schema.

    Returns
    -----
    data: Data format after any processing.

    """
    data = deserialize(data, input_schema_path)

    # Add further data preprocessing if needed
    return data

def post_inference(yhat):
    """

```

(continues on next page)

(continued from previous page)

```

    Post-process the model results.

    Parameters
    -----
    yhat: Data format after calling model.predict.

    Returns
    -----
    yhat: Data format after any processing.

    """

    return yhat.numpy().tolist()

def predict(data, model=load_model(), input_schema_path=os.path.join(os.path.dirname(os.
↪path.realpath(__file__)), "input_schema.json")):
    """
    Returns prediction given the model and data to predict.

    Parameters
    -----
    model: Model instance returned by load_model API
    data: Data format as expected by the predict API of the core estimator.
    input_schema_path: path of input schema.

    Returns
    -----
    predictions: Output from scoring server
        Format: {'prediction': output from model.predict method}

    """
    inputs = pre_inference(data, input_schema_path)

    yhat = post_inference(
        model(inputs)
    )
    return {'prediction': yhat}

```

Step 2: Update post\_inference method in score.py to find the index corresponding the maximum value and return. We can use argmax function from tensorflow to achieve that. Here is the modified code -

```

1  import os
2  import sys
3  from functools import lru_cache
4  import pandas as pd
5  import numpy as np
6  import json
7  import tensorflow as tf
8  from io import BytesIO
9  import base64
10
11 model_name = 'model.h5'

```

(continues on next page)

(continued from previous page)

```

12
13
14 """
15 Inference script. This script is used for prediction by scoring server when schema is_
16 ↪known.
17 """
18
19 @lru_cache(maxsize=10)
20 def load_model(model_file_name=model_name):
21     """
22     Loads model from the serialized format
23
24     Returns
25     -----
26     model: a model instance on which predict API can be invoked
27     """
28     model_dir = os.path.dirname(os.path.realpath(__file__))
29     if model_dir not in sys.path:
30         sys.path.insert(0, model_dir)
31     contents = os.listdir(model_dir)
32     if model_file_name in contents:
33         print(f'Start loading {model_file_name} from model directory {model_dir} ...')
34         loaded_model = tf.keras.models.load_model(os.path.join(model_dir, model_file_
35 ↪name))
36
37         print("Model is successfully loaded.")
38         return loaded_model
39     else:
40         raise Exception(f'{model_file_name} is not found in model directory {model_dir}')
41
42 @lru_cache(maxsize=1)
43 def fetch_data_type_from_schema(input_schema_path=os.path.join(os.path.dirname(os.path.
44 ↪realpath(__file__)), "input_schema.json")):
45     """
46     Returns data type information fetch from input_schema.json.
47
48     Parameters
49     -----
50     input_schema_path: path of input schema.
51
52     Returns
53     -----
54     data_type: data type fetch from input_schema.json.
55
56     """
57     data_type = {}
58     if os.path.exists(input_schema_path):
59         schema = json.load(open(input_schema_path))
60         for col in schema['schema']:
61             data_type[col['name']] = col['dtype']

```

(continues on next page)

(continued from previous page)

```

61     else:
62         print("input_schema has to be passed in in order to recover the same data type.↳
↳pass `X_sample` in `ads.model.framework.tensorflow_model.TensorFlowModel.prepare`↳
↳function to generate the input_schema. Otherwise, the data type might be changed after↳
↳serialization/deserialization.")
63     return data_type
64
65
66 def deserialize(data, input_schema_path):
67     """
68     Deserialize json-serialized data to data in original type when sent to
69     predict.
70
71     Parameters
72     -----
73     data: serialized input data.
74     input_schema_path: path of input schema.
75
76     Returns
77     -----
78     data: deserialized input data.
79
80     """
81     data_type = data.get('data_type', '')
82     json_data = data.get('data', data)
83
84     if "numpy.ndarray" in data_type:
85         load_bytes = BytesIO(base64.b64decode(json_data.encode('utf-8')))
86         return np.load(load_bytes, allow_pickle=True)
87     if "pandas.core.series.Series" in data_type:
88         return pd.Series(json_data)
89     if "pandas.core.frame.DataFrame" in data_type:
90         return pd.read_json(json_data, dtype=fetch_data_type_from_schema(input_schema_
↳path))
91     if "tensorflow.python.framework.ops.EagerTensor" in data_type:
92         load_bytes = BytesIO(base64.b64decode(json_data.encode('utf-8')))
93         return tf.convert_to_tensor(np.load(load_bytes, allow_pickle=True))
94
95     return json_data
96
97 def pre_inference(data, input_schema_path):
98     """
99     Preprocess json-serialized data to feed into predict function.
100
101     Parameters
102     -----
103     data: Data format as expected by the predict API of the core estimator.
104     input_schema_path: path of input schema.
105
106     Returns
107     -----
108     data: Data format after any processing.

```

(continues on next page)

(continued from previous page)

```

109     """
110     data = deserialize(data, input_schema_path)
111
112     # Add further data preprocessing if needed
113     return data
114
115 def post_inference(yhat):
116     """
117     Post-process the model results.
118
119     Parameters
120     -----
121     yhat: Data format after calling model.predict.
122
123     Returns
124     -----
125     yhat: Data format after any processing.
126
127     """
128     yhat = tf.argmax(yhat, axis=1) # Get the index of the max value
129     return yhat.numpy().tolist()
130
131 def predict(data, model=load_model(), input_schema_path=os.path.join(os.path.dirname(os.
132     ↪path.realpath(__file__)), "input_schema.json")):
133     """
134     Returns prediction given the model and data to predict.
135
136     Parameters
137     -----
138     model: Model instance returned by load_model API
139     data: Data format as expected by the predict API of the core estimator.
140     input_schema_path: path of input schema.
141
142     Returns
143     -----
144     predictions: Output from scoring server
145     Format: {'prediction': output from model.predict method}
146
147     """
148     inputs = pre_inference(data, input_schema_path)
149
150     yhat = post_inference(
151         model(inputs)
152     )
153     return {'prediction': yhat}

```

Step 3: Verify the changes

```
print(tensorflow_model.verify(testx[:3])['prediction'])
```

Start loading model.h5 from model directory /tmp/tmppkco6xrt ...  
Model is successfully loaded.

(continues on next page)

(continued from previous page)

```
[7, 2, 1]
```

Step 4: Register the model

```
model_id = tensorflow_model.save()
```

Step 5: Deploy and generate the endpoint

```
>>> # Deploy and create an endpoint for the TensorFlow model
>>> tensorflow_model.deploy(
    display_name="TensorFlow Model For Classification",
    deployment_log_group_id = "ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id = "ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id = "ocid1.log.oc1.xxx.xxxxx"
)
>>> print(f"Endpoint: {tensorflow_model.model_deployment.url}")
https://modeldeployment.{region}.oci.customer-oci.com/ocid1.datasciencemodeldeployment.
↳ oc1.xxx.xxxxx
```

Step 6: Run prediction from the endpoint

```
print(tensorflow_model.predict(testx[:3])['prediction'])
```

```
[7, 2, 1]
```

## 12.3 Deploying model

Once you have ADS Model object, you can call deploy function to deploy the model and generate the endpoint.

Here is an example of deploying LightGBM model:

```
import lightgbm as lgb
import tempfile
from ads.common.model_metadata import UseCaseType
from ads.model.framework.lightgbm_model import LightGBMModel
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset and Prepare train and test split
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Train a LightGBM Classifier model
train = lgb.Dataset(X_train, label=y_train)
param = {
    'objective': 'multiclass', 'num_class': 3,
}
lightgbm_estimator = lgb.train(param, train)

# Instantite ads.model.LightGBMModel using the trained LGBM Model
```

(continues on next page)

(continued from previous page)

```

lightgbm_model = LightGBMModel(estimator=lightgbm_estimator, artifact_dir=tempfile.
↳mkdtemp())

# Autogenerate score.py, pickled model, runtime.yaml, input_schema.json and output_
↳schema.json
lightgbm_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    X_sample=X_train,
    y_sample=y_train,
    use_case_type=UseCaseType.BINARY_CLASSIFICATION,
)

# Verify generated artifacts
lightgbm_model.verify(X_test)

# Register LightGBM model
model_id = lightgbm_model.save()

# Deploy LightGBM model
lightgbm_model.deploy(
    display_name="LightGBM Model",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)

# Get endpoint of deployed model
model_deployment_url = lightgbm_model.model_deployment.url

# Generate prediction by invoking the deployed endpoint
lightgbm_model.predict(X_test)["prediction"]

```

Here example retrieve predictions from model deployment endpoint using oci-cli:

```

export model_deployment_url=<model_deployment_url>/predict
oci raw-request --http-method POST \
  --target-uri $model_deployment_url \
  --request-body '{"data": [[5.6, 2.7, 4.2, 1.3]]}'

```

Find more information about oci raw-request command [here](#).

### 12.3.1 Deploy

You can use the `.deploy()` method to deploy a model. You must first save the model to the model catalog, and then deploy it.

The `.deploy()` method returns a `ModelDeployment` object. Specify deployment attributes such as display name, instance type, number of instances, maximum router bandwidth, and logging groups. The API takes the following parameters:

See [API documentation](#) for more details about the parameters.

---

#### Tips



- Providing `deployment_access_log_id` and `deployment_predict_log_id` helps in debugging your model inference setup.
- Default Load Balancer configuration has bandwidth of 10 Mbps. [Refer service document to help you choose the right setup.](#)
- Check for supported instance shapes [here](#) .

## 12.3.2 Predict

To invoke the endpoint of your deployed model, call the `.predict()` method. The `.predict()` method sends a request to the deployed endpoint, and computes the inference values based on the data that you input in the `.predict()` method.

See how to deploy and invoke deployed endpoint for different frameworks [here](#).

See [API documentation](#) for more details about the parameters.

## 12.3.3 Observability

tail or head logs generated by the model deployment instances -

```
lightgbm_model.model_deployment.logs().tail()
```

## 12.3.4 Update Model Deployment

You can update the existing Model Deployment by using `.update_deployment()` method. See [API documentation](#) for more details.

```
lightgbm_model.update_deployment(
    properties=ModelDeploymentProperties(
        access_log_id="ocid1.log.oc1.xxx.xxxxx",
        description="Description for Custom Model",
        freeform_tags={"key": "value"},
    )
    wait_for_completion = True,
)
```

# 12.4 Loading model

## 12.4.1 Load Registered Model

Load and recreate *framework specific wrapper* objects using the `ocid` value of your model.

The loaded artifact can be used for running inference in local environment. You can update the artifact files to change your `score.py` or model and then register as a new model. See [here](#) to learn how to change `score.py`

Here is an example for loading back a LightGBM model that was previously registered. See [API documentation](#) for more details.

```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_model_catalog(
    "ocid1.datasciencemodel.oc1.xxx.xxxxx",
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
)
```

You can print a model to see some details about it.

```
print(lgbm_model)
```

```
algorithm: null
artifact_dir: lgbm-download-test:
-   - model.pkl
    - output_schema.json
    - runtime.yaml
    - score.py
    - input_schema.json
framework: null
model_deployment_id: null
model_id: ocid1.datasciencemodel.oc1.iad.xxx
```

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's Step column displays a Status of Done for the initiate step. And the Details column explains what the initiate step did such as generating a `score.py` file. The Step column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The Status column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

New in version 2.6.9.

Alternatively the `.from_id()` method can be used to load a model. In future releases, the `.from_model_catalog()` method will be deprecated and replaced with the `from_id()`. See [API documentation](#) for more details.

```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_id(
    "ocid1.datasciencemodel.oc1.xxx.xxxxx",
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
    bucket_uri=<oci://<bucket_name>@<namespace>/prefix/>,
    force_overwrite=True,
    remove_existing_artifact=True,
)
```

		Actions Needed
Step	Status	Details
initiate	Done	Initiated the model
prepare()	Available	Generated runtime.yaml
		Generated score.py
		Serialized model
		Populated metadata(Custom, Taxonomy and Provenance)
verify()	Not Available	Local tested .predict from score.py
save()	Not Available	Conducted Introspect Test
		Uploaded artifact to model catalog
deploy()	Not Available	Deployed the model
predict()	Not Available	Called deployment predict endpoint

## 12.4.2 Load Deployed Model

Load and recreate *framework specific wrapper* objects using the `ocid` value of your OCI Model Deployment instance.

The loaded artifact can be used for running inference in local environment. You can update the artifact files to change your `score.py` or model and then register as a new model. See [here](#) to learn how to change `score.py`

Here is an example for loading back a LightGBM model that was previously deployed. See [API doc](#) for more information.

```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_model_deployment(
    "ocid1.datasciencemodel.oc1.xxx.xxxxx",
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
)
```

You can print a model to see some details about it.

```
print(lgbm_model)
```

```
algorithm: null
artifact_dir: lgbm-download-test:
-   - model.pkl
    - output_schema.json
    - runtime.yaml
    - score.py
    - input_schema.json
framework: null
model_deployment_id: null
model_id: ocid1.datasciencemodel.oc1.iad.xxx
```

New in version 2.6.9.

Alternatively the `.from_id()` method can be used to load a model from the Model Deployment. In future releases, the `.from_model_deployment()` method will be deprecated and replaced with the `from_id()`. See [API documentation](#) for more details.

```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_id(
    "ocid1.datasciencemodeldeployment.oc1.xxx",
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
    bucket_uri=<oci://<bucket_name>@<namespace>/prefix/>,
    force_overwrite=True,
    remove_existing_artifact=True,
)
```

### 12.4.3 Load Model From Object Storage

Load and recreate *framework specific wrapper* objects from the existing model artifact archive.

The loaded artifact can be used for running inference in local environment. You can update the artifact files to change your `score.py` or model and then register as a new model. See [here](#) to learn how to change `score.py`

Here is an example for loading back a LightGBM model that was previously saved to the Object Storage. See [API doc](#) for more information.

```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_model_artifact(
    <oci://<bucket_name>@<namespace>/prefix/lgbm_model_artifact.zip>,
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
    force_overwrite=True
)
```

A model loaded from an artifact archive can be registered and deployed.

You can print a model to see some details about it.

```
print(lgbm_model)
```

```
algorithm: null
artifact_dir: lgbm-download-test:
-   - model.pkl
    - output_schema.json
    - runtime.yaml
    - score.py
    - input_schema.json
framework: null
model_deployment_id: null
model_id: ocid1.datasciencemodel.oc1.iad.xxx
```

## 12.4.4 Large Model Artifacts

New in version 2.6.4.

Large models are models with artifacts between 2 and 6 GB. You must first download large models from the model catalog to an Object Storage bucket, and then transfer them to local storage. For model artifacts that are less than 2 GB, you can use the same approach, or download them directly to local storage. An Object Storage bucket is required with Data Science service access granted to that bucket.

If you don't have an Object Storage bucket, create one using the OCI SDK or the Console. Create an [Object Storage bucket](#). Make a note of the namespace, compartment, and bucket name. Configure the following policies to allow the Data Science service to read and write the model artifact to the Object Storage bucket in your tenancy. An administrator must configure these policies in [IAM](#) in the Console.

```
Allow service datascience to manage object-family in compartment <compartment> where ALL
↪ {target.bucket.name='<bucket_name>'}
```

```
Allow service objectstorage to manage object-family in compartment <compartment> where
↪ ALL {target.bucket.name='<bucket_name>'}
```

The following example loads a model using the large model artifact approach. The `bucket_uri` has the following syntax: `oci://<bucket_name>@<namespace>/<path>/` See [API documentation](#) for more details.

```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_model_catalog(
    "ocid1.datasciencemodel.oc1.xxx.xxxxx",
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
    bucket_uri=<oci://<bucket_name>@<namespace>/prefix/>,
    force_overwrite=True,
    remove_existing_artifact=True,
)
```

Here is an example for loading back a LightGBM model with large artifact from Model Deployment. See [API doc](#) for more information.

```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_model_deployment(
    "ocid1.datasciencemodel.oc1.xxx.xxxxx",
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
    bucket_uri=<oci://<bucket_name>@<namespace>/prefix/>,
    force_overwrite=True,
    remove_existing_artifact=True,
)
```

New in version 2.6.9.

Alternatively the `.from_id()` method can be used to load registered or deployed model. In future releases, the `.from_model_catalog()` and `.from_model_deployment()` methods will be deprecated and replaced with the `from_id()`. See [API documentation](#) for more details.

```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_id(
    "ocid1.datasciencemodel.oc1.xxx.xxxxx",
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
    bucket_uri=<oci://<bucket_name>@<namespace>/prefix/>,
    force_overwrite=True,
    remove_existing_artifact=True,
)
```

## 12.5 Frameworks

New in version 2.5.9.

### 12.5.1 SklearnModel

See [API Documentation](#)

#### 12.5.1.1 Overview

The `SklearnModel` class in ADS is designed to allow you to rapidly get a Scikit-learn model into production. The `.prepare()` method creates the model artifacts that are needed to deploy a functioning model without you having to configure it or write code. However, you can customize the required `score.py` file.

The `.verify()` method simulates a model deployment by calling the `load_model()` and `predict()` methods in the `score.py` file. With the `.verify()` method, you can debug your `score.py` file without deploying any models. The `.save()` method deploys a model artifact to the model catalog. The `.deploy()` method deploys a model to a REST endpoint.

The following steps take your trained `scikit-learn` model and deploy it into production with a few lines of code.

#### Create a Scikit-learn Model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

seed = 42

X, y = make_classification(n_samples=10000, n_features=15, n_classes=2, flip_y=0.05)
trainx, testx, trainy, testy = train_test_split(X, y, test_size=30, random_state=seed)
model = RandomForestClassifier(
    n_estimators=100, random_state=42
)
model.fit(
    trainx,
    trainy,
)
```

### 12.5.1.2 Prepare Model Artifact

```
from ads.model.framework.sklearn_model import SklearnModel
from ads.common.model_metadata import UseCaseType

sklearn_model = SklearnModel(estimator=model, artifact_dir="~/sklearn_artifact_dir")
sklearn_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    training_conda_env="generalml_p38_cpu_v1",
    X_sample=trainx,
    y_sample=trainy,
    use_case_type=UseCaseType.BINARY_CLASSIFICATION,
)
```

Instantiate a `ads.model.framework.sklearn_model.SklearnModel()` object with an Scikit-learn model. Each instance accepts the following parameters:

- `artifact_dir`: `str`: Artifact directory to store the files needed for deployment.
- `auth`: (`Dict`, optional): Defaults to `None`. The default authentication is set using the `ads.set_auth` API. To override the default, use `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()` and create the appropriate authentication signer and the `**kwargs` required to instantiate the `IdentityClient` object.
- `estimator`: (`Callable`): Trained Scikit-learn model or Scikit-learn pipeline.
- `properties`: (`ModelProperties`, optional): Defaults to `None`. The `ModelProperties` object required to save and deploy a model.

The `properties` is an instance of the `ModelProperties` class and has the following predefined fields:

- `bucket_uri`: `str`
- `compartment_id`: `str`
- `deployment_access_log_id`: `str`
- `deployment_bandwidth_mbps`: `int`
- `deployment_instance_count`: `int`
- `deployment_instance_shape`: `str`
- `deployment_log_group_id`: `str`
- `deployment_predict_log_id`: `str`
- `deployment_memory_in_gbs`: `Union[float, int]`
- `deployment_ocpus`: `Union[float, int]`
- `inference_conda_env`: `str`
- `inference_python_version`: `str`
- `overwrite_existing_artifact`: `bool`
- `project_id`: `str`
- `remove_existing_artifact`: `bool`
- `training_conda_env`: `str`
- `training_id`: `str`

- `training_python_version`: str
- `training_resource_id`: str
- `training_script_path`: str

By default, `properties` is populated from the environment variables when not specified. For example, in notebook sessions the environment variables are preset and stored in project id (`PROJECT_OCID`) and compartment id (`NB_SESSION_COMPARTMENT_OCID`). So `properties` populates these environment variables, and uses the values in methods such as `.save()` and `.deploy()`. Pass in values to overwrite the defaults. When you use a method that includes an instance of `properties`, then `properties` records the values that you pass in. For example, when you pass `inference_conda_env` into the `.prepare()` method, then `properties` records the value. To reuse the `properties` file in different places, you can export the `properties` file using the `.to_yaml()` method then reload it into a different machine using the `.from_yaml()` method.

### 12.5.1.3 Summary Status

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's Step column displays a Status of Done for the initiate step. And the Details column explains what the initiate step did such as generating a `score.py` file. The Step column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The Status column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

		Actions Needed
Step	Status	Details
initiate	Done	Initiated the model
prepare()	Available	Generated runtime.yaml
		Generated score.py
		Serialized model
		Populated metadata(Custom, Taxonomy and Provenance)
verify()	Not Available	Local tested .predict from score.py
save()	Not Available	Conducted Introspect Test
		Uploaded artifact to model catalog
deploy()	Not Available	Deployed the model
predict()	Not Available	Called deployment predict endpoint



### 12.5.1.4 Register Model

```
>>> # Register the model
>>> model_id = sklearn_model.save()

Start loading model.joblib from model directory /tmp/tmp9l0uhtbb ...
Model is successfully loaded.
['output_schema.json', 'runtime.yaml', 'model.joblib', 'score.py', 'input_schema.json']

'ocid1.datasciencemodel.oc1.xxx.xxxxx'
```

### 12.5.1.5 Deploy and Generate Endpoint

```
>>> # Deploy and create an endpoint for the Random Forest model
>>> sklearn_model.deploy(
    display_name="Random Forest Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)
>>> print(f"Endpoint: {sklearn_model.model_deployment.url}")
https://modeldeployment.{region}.oci.customer-oci.com/ocid1.datasciencemodeldeployment.
oc1.xxx.xxxxx
```

### 12.5.1.6 Run Prediction against Endpoint

```
>>> # Generate prediction by invoking the deployed endpoint
>>> sklearn_model.predict(testx)['prediction']
[1,0,...,1]
```

### 12.5.1.7 Run Prediction with oci raw-request command

Model deployment endpoints can be invoked with the OCI-CLI. The below examples invoke a model deployment with the CLI with different types of payload: `json`, `numpy.ndarray`, `pandas.core.frame.DataFrame` or `dict`.

#### 12.5.1.7.1 json payload example

```
>>> # Prepare data sample for prediction
>>> data = testx[[10]]
>>> data
array([[ 0.41330051,  0.67658927, -0.39189561,  0.21879805, -0.79208514,
         0.0906022 , -1.60595137,  1.65853693, -1.61337437,  0.82302124,
        -0.87032051,  0.70721209, -1.81956653, -0.26537296, -0.25471684]])
```

Use output of the data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↳datasciencemodeldeployment.oc1.xxx.xxxxx/predict
export data='{"data": [[ 0.41330051, 0.67658927, ... , -0.25471684]]}'
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"
```

### 12.5.1.7.2 *numpy.ndarray* payload example

```
>>> # Prepare data sample for prediction
>>> from io import BytesIO
>>> import base64
>>> import numpy as np

>>> data = testx[[10]]
>>> np_bytes = BytesIO()
>>> np.save(np_bytes, data, allow_pickle=True)
>>> data = base64.b64encode(np_bytes.getvalue()).decode("utf-8")
>>> print(data)
k05VTVBZAB2AHsnZGVzY.....4UdN0L8=
```

Use printed output of base64 data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↳datasciencemodeldeployment.oc1.xxx.xxxxx/predict
export data='{"data": "k05VTVBZAB2AHsnZGVzY.....4UdN0L8=", "data_type": "numpy.ndarray"}'
↳
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"
```

### 12.5.1.7.3 *pandas.core.frame.DataFrame* payload example

```
>>> # Prepare data sample for prediction
>>> import pandas as pd

>>> df = pd.DataFrame(testx[[10]])
>>> print(json.dumps(df.to_json()))
{"0":{"0":{"0":0.4133005141},"1":{"0":0.676589266},...,"14":{"0":-0.2547168443}}"
```

Use printed output of DataFrame data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↳datasciencemodeldeployment.oc1.xxx.xxxxx/predict
export data='{"data": "{\"0\":{\"0\":0.4133005141},...,\"14\":{\"0\":-0.2547168443}}",
↳"data_type": "pandas.core.frame.DataFrame"}'
oci raw-request \
```

(continues on next page)

(continued from previous page)

```
--http-method POST \
--target-uri $uri \
--request-body "$data"
```

#### 12.5.1.7.4 *dict* payload example

```
>>> # Prepare data sample for prediction
>>> import pandas as pd
```

```
>>> df = pd.DataFrame(testx[[10]])
>>> print(json.dumps(df.to_dict()))
{"0": {"0": 0.413300514080485}, "1": {"0": 0.6765892660311731}, ..., "14": {"0": -0.
↪ 2547168443271222}}
```

Use printed output of *dict* data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↪ datasciencemodeldeployment.oc1.xxx.xxxx/predict
export data='{"data": {"0": {"0": 0.413300514080485}, ..., "14": {"0": -0.
↪ 2547168443271222}}}'
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"
```

#### 12.5.1.7.5 Expected output of raw-request command

```
{
  "data": {
    "prediction": [
      0
    ]
  },
  "headers": {
    "Connection": "keep-alive",
    "Content-Length": "18",
    "Content-Type": "application/json",
    "Date": "Wed, 07 Dec 2022 18:31:39 GMT",
    "X-Content-Type-Options": "nosniff",
    "opc-request-id": "E1125E17AE084DFAB6BCCFA045C16966/0BBB65235292EE0817B67BD9141A620A/
↪ 5956FE428CBAB2878EBA605CEECAD39D",
    "server": "unicorn"
  },
  "status": "200 OK"
}
```

## 12.5.1.8 Examples

```

from ads.model.framework.sklearn_model import SklearnModel
from ads.common.model_metadata import UseCaseType

from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

import tempfile

seed = 42

# Create a classification dataset
X, y = make_classification(n_samples=10000, n_features=15, n_classes=2, flip_y=0.05)
trainx, testx, trainy, testy = train_test_split(X, y, test_size=30, random_state=seed)

# Train LGBM model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(
    trainx,
    trainy,
)

# Deploy the model, test it and clean up.
# Prepare Model Artifact for RandomForest Classifier model
artifact_dir = tempfile.mkdtemp()
sklearn_model = SklearnModel(estimator=model, artifact_dir=artifact_dir)
sklearn_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    training_conda_env="generalml_p38_cpu_v1",
    use_case_type=UseCaseType.BINARY_CLASSIFICATION,
    X_sample=trainx,
    y_sample=trainy,
    force_overwrite=True,
)

# Check if the artifacts are generated correctly.
# The verify method invokes the ``predict`` function defined inside ``score.py`` in the
↳ artifact_dir

sklearn_model.verify(testx[:10])["prediction"]
sklearn_model.save(display_name="SKLearn Model")

# Deploy and create an endpoint for the RandomForest model
sklearn_model.deploy(
    display_name="Random Forest Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxx",
)

```

(continues on next page)

(continued from previous page)

```
print(f"Endpoint: {sklearn_model.model_deployment.url}")

sklearn_model.predict(testx)["prediction"]

# To delete the deployed endpoint uncomment the following line
# sklearn_model.delete_deployment(wait_for_completion=True)
```

## 12.5.2 PyTorchModel

See [API Documentation](#)

### 12.5.2.1 Overview

The `ads.model.framework.pytorch_model.PyTorchModel` class in ADS is designed to allow you to rapidly get a PyTorch model into production. The `.prepare()` method creates the model artifacts that are needed to deploy a functioning model without you having to configure it or write code. However, you can customize the required `score.py` file.

The `.verify()` method simulates a model deployment by calling the `load_model()` and `predict()` methods in the `score.py` file. With the `.verify()` method, you can debug your `score.py` file without deploying any models. The `.save()` method deploys a model artifact to the model catalog. The `.deploy()` method deploys a model to a REST endpoint.

The following steps take your trained PyTorch model and deploy it into production with a few lines of code.

#### Create a PyTorch Model

Load a [ResNet18](#) model and put it into evaluation mode.

```
import torch
import torchvision

model = torchvision.models.resnet18(pretrained=True)
model.eval()
```

### 12.5.2.2 Prepare Model Artifact

#### 12.5.2.2.1 Save as TorchScript

New in version 2.6.9.

Serializing model in TorchScript program by setting `use_torch_script` to `True`, you can load the model and run inference without defining the model class.

```
from ads.common.model_metadata import UseCaseType
from ads.model.framework.pytorch_model import PyTorchModel

import tempfile

# Prepare the model
```

(continues on next page)

(continued from previous page)

```

artifact_dir = "pytorch_model_artifact"
pytorch_model = PyTorchModel(model, artifact_dir=artifact_dir)
pytorch_model.prepare(
    inference_conda_env="pytorch110_p38_cpu_v1",
    training_conda_env="pytorch110_p38_cpu_v1",
    use_case_type=UseCaseType.IMAGE_CLASSIFICATION,
    force_overwrite=True,
    use_torch_script=True,
)
# You don't need to modify the score.py generated. The model can be loaded without
# defining the model class.
# More info here - https://pytorch.org/tutorials/beginner/saving\_loading\_models.html
# export-load-model-in-torchscript-format

```

#### 12.5.2.2.2 Save state\_dict

```

from ads.common.model_metadata import UseCaseType
from ads.model.framework.pytorch_model import PyTorchModel

import tempfile

# Prepare the model
artifact_dir = "pytorch_model_artifact"
pytorch_model = PyTorchModel(model, artifact_dir=artifact_dir)
pytorch_model.prepare(
    inference_conda_env="pytorch110_p38_cpu_v1",
    training_conda_env="pytorch110_p38_cpu_v1",
    use_case_type=UseCaseType.IMAGE_CLASSIFICATION,
    force_overwrite=True,
)

# The score.py generated requires you to create the class instance of the Model before
# the weights are loaded.
# More info here - https://pytorch.org/tutorials/beginner/saving\_loading\_models.html
# save-load-state-dict-recommended

```

Open `pytorch_model_artifact/score.py` and edit the code to instantiate the model class. The edits are highlighted -

```

import os
import sys
from functools import lru_cache
import torch
import json
from typing import Dict, List
import numpy as np
import pandas as pd
from io import BytesIO
import base64
import logging

```

(continues on next page)

(continued from previous page)

```

import torchvision
the_model = torchvision.models.resnet18()

model_name = 'model.pt'

"""
Inference script. This script is used for prediction by scoring server when schema is
↪known.
"""

@lru_cache(maxsize=10)
def load_model(model_file_name=model_name):
    """
    Loads model from the serialized format

    Returns
    -----
    model: a model instance on which predict API can be invoked
    """
    model_dir = os.path.dirname(os.path.realpath(__file__))
    if model_dir not in sys.path:
        sys.path.insert(0, model_dir)
    contents = os.listdir(model_dir)
    if model_file_name in contents:
        print(f'Start loading {model_file_name} from model directory {model_dir} ...')
        model_state_dict = torch.load(os.path.join(model_dir, model_file_name))
        print(f"loading {model_file_name} is complete.")
    else:
        raise Exception(f'{model_file_name} is not found in model directory {model_dir}')

    # User would need to provide reference to the TheModelClass and
    # construct the the_model instance first before loading the parameters.
    # the_model = TheModelClass(*args, **kwargs)
    try:
        the_model.load_state_dict(model_state_dict)
    except NameError as e:
        raise NotImplementedError("TheModelClass instance must be constructed before
↪loading the parameters. Please modify the load_model() function in score.py." )
    except Exception as e:
        raise e

    the_model.eval()
    print("Model is successfully loaded.")

    return the_model

```

Instantiate a `PyTorchModel()` object with a PyTorch model. Each instance accepts the following parameters:

- `artifact_dir`: str. Artifact directory to store the files needed for deployment.
- `auth`: (Dict, optional): Defaults to None. The default authentication is set using the ads.

`set_auth` API. To override the default, use `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()` and create the appropriate authentication signer and the `**kwargs` required to instantiate the `IdentityClient` object.

- `estimator`: Callable. Any model object generated by the PyTorch framework.
- `properties`: (`ModelProperties`, optional). Defaults to `None`. The `ModelProperties` object required to save and deploy model.

The `properties` is an instance of the `ModelProperties` class and has the following predefined fields:

- `bucket_uri`: str
- `compartment_id`: str
- `deployment_access_log_id`: str
- `deployment_bandwidth_mbps`: int
- `deployment_instance_count`: int
- `deployment_instance_shape`: str
- `deployment_log_group_id`: str
- `deployment_predict_log_id`: str
- `deployment_memory_in_gbs`: Union[float, int]
- `deployment_ocpus`: Union[float, int]
- `inference_conda_env`: str
- `inference_python_version`: str
- `overwrite_existing_artifact`: bool
- `project_id`: str
- `remove_existing_artifact`: bool
- `training_conda_env`: str
- `training_id`: str
- `training_python_version`: str
- `training_resource_id`: str
- `training_script_path`: str

By default, `properties` is populated from the environment variables when not specified. For example, in notebook sessions the environment variables are preset and stored in project id (`PROJECT_OCID`) and compartment id (`NB_SESSION_COMPARTMENT_OCID`). So `properties` populates these environment variables, and uses the values in methods such as `.save()` and `.deploy()`. Pass in values to overwrite the defaults. When you use a method that includes an instance of `properties`, then `properties` records the values that you pass in. For example, when you pass `inference_conda_env` into the `.prepare()` method, then `properties` records the value. To reuse the `properties` file in different places, you can export the `properties` file using the `.to_yaml()` method then reload it into a different machine using the `.from_yaml()` method.



### 12.5.2.3 Verify Changes to Score.py

Download and load an image for prediction

```
# Download an image
import urllib.request
url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog.jpg")
try: urllib.URLopener().retrieve(url, filename)
except: urllib.request.urlretrieve(url, filename)

# Preprocess the image and convert to torch.Tensor
from PIL import Image
from torchvision import transforms
input_image = Image.open(filename)
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by the model
```

Verify score.py changes by running inference locally

```
>>> prediction = pytorch_model.verify(input_batch)["prediction"]
>>> import numpy as np
>>> np.argmax(prediction)
258
```

### 12.5.2.4 Summary Status

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's Step column displays a Status of Done for the initiate step. And the Details column explains what the initiate step did such as generating a `score.py` file. The Step column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The Status column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

		Actions Needed
Step	Status	Details
initiate	Done	Initiated the model
prepare()	Available	Generated runtime.yaml
		Generated score.py
		Serialized model
		Populated metadata(Custom, Taxonomy and Provenance)
verify()	Not Available	Local tested .predict from score.py
save()	Not Available	Conducted Introspect Test
		Uploaded artifact to model catalog
deploy()	Not Available	Deployed the model
predict()	Not Available	Called deployment predict endpoint

#### 12.5.2.5 Register Model

```
>>> # Register the model
>>> model_id = pytorch_model.save()

Start loading model.pt from model directory /tmp/tmpf11gnx9c ...
loading model.pt is complete.
Model is successfully loaded.
['.score.py.swp', 'score.py', 'model.pt', 'runtime.yaml']

'ocid1.datasciencemodel.oc1.xxx.xxxxx'
```

#### 12.5.2.6 Deploy and Generate Endpoint

```
>>> # Deploy and create an endpoint for the TensorFlow model
>>> pytorch_model.deploy(
    display_name="PyTorch Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)

>>> print(f"Endpoint: {pytorch_model.model_deployment.url}")

https://modeldeployment.{region}.oci.customer-oci.com/ocid1.datasciencemodeldeployment.
↳ oc1.xxx.xxxxx
```

### 12.5.2.7 Run Prediction against Endpoint

```
# Download an image
import urllib.request
url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog.jpg")
try: urllib.URLopener().retrieve(url, filename)
except: urllib.request.urlretrieve(url, filename)

# Preprocess the image and convert to torch.Tensor
from PIL import Image
from torchvision import transforms
input_image = Image.open(filename)
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by the model

# Generate prediction by invoking the deployed endpoint
prediction = pytorch_model.predict(input_batch)['prediction']
print(np.argmax(prediction))
```

258

#### 12.5.2.7.1 Predict with Image

New in version 2.6.7.

Predict Image by passing a uri, which can be http(s), local path, or other URLs (e.g. starting with “oci://”, “s3://”, and “gcs://”), of the image or a PIL.Image.Image object using the `image` argument in `predict()` to predict a single image. The image will be converted to a tensor and then serialized so it can be passed to the endpoint. You can catch the tensor in `score.py` to perform further transformation.

**Note:** The payload size limit is 10 MB. Read more about invoking a model deployment [here](#).

Given the size limitation, the example below is with resized image. To pass an image and invoke prediction, additional code inside `score.py` is required to preprocess the data. Open `pytorch_model_artifact/score.py` and update the `pre_inference()` method. The edits are highlighted:

```
def pre_inference(data, input_schema_path):
    """
    Preprocess json-serialized data to feed into predict function.

    Parameters
    -----
    data: Data format as expected by the predict API of the core estimator.
    input_schema_path: path of input schema.
```

(continues on next page)

(continued from previous page)

```

Returns
-----
data: Data format after any processing.
"""
data = deserialize(data, input_schema_path)
import torchvision.transforms as transforms
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    ),
])
input_tensor = preprocess(data)
input_batch = input_tensor.unsqueeze(0)
return input_batch

```

Save `score.py` and verify prediction works:

```

>>> uri = ("https://github.com/oracle-samples/oci-data-science-ai-samples/tree/master/
↳model_deploy_examples/images/dog_resized.jpg")
>>> prediction = pytorch_model.verify(image=uri)["prediction"]
>>> import numpy as np
>>> np.argmax(prediction)
258

```

Re-deploy model with updated `score.py`:

```

pytorch_model.deploy(
    display_name="PyTorch Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxxx",
)

```

Run prediction with the image provided:

```

uri = ("https://github.com/oracle-samples/oci-data-science-ai-samples/tree/master/model_
↳deploy_examples/images/dog_resized.jpg")

# Generate prediction by invoking the deployed endpoint
prediction = pytorch_model.predict(image=uri)['prediction']

```

### 12.5.2.8 Run Prediction with oci raw-request command

Model deployment endpoints can be invoked with the OCI-CLI. This example invokes a model deployment with the CLI with a `torch.Tensor` payload:

#### 12.5.2.8.1 *torch.Tensor* payload example

```
>>> # Prepare data sample for prediction and save it to file 'data-payload'
>>> from io import BytesIO
>>> import base64

>>> buffer = BytesIO()
>>> torch.save(input_batch, buffer)
>>> data = base64.b64encode(buffer.getvalue()).decode("utf-8")
>>> with open('data-payload', 'w') as f:
>>>     f.write('{"data": "' + data + '", "data_type": "torch.Tensor"}')
```

File `data-payload` will have this information:

```
{"data": "UESDBAAACAgAAAAAAAAAAAAAAAAAAAAAAQ .....
.....
...AAAAEAAABQSwUGAAAAAAAAAAwC3AAAA0jEJAAAA", "data_type": "torch.Tensor"}
```

Use file `data-payload` with `data` and `endpoint` to invoke prediction with `raw-request` command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
datasciencemodeldeployment.oc1.xxx.xxxx/predict
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body file://data-payload
```

#### 12.5.2.8.2 Expected output of raw-request command

```
{
  "data": {
    "prediction": [
      [
        0.0159152802079916,
        -1.5496551990509033,
        .....
        2.5116958618164062
      ]
    ]
  },
  "headers": {
    "Connection": "keep-alive",
    "Content-Length": "19398",
    "Content-Type": "application/json",
    "Date": "Thu, 08 Dec 2022 18:28:41 GMT",
```

(continues on next page)

(continued from previous page)

```

    "X-Content-Type-Options": "nosniff",
    "opc-request-id": "BD80D931A6EA4C718636ECE00730B255/86111E71C1B33C24988C59C27F15ECDE/
    ↪E94BBB27AC3F48CB68F41135073FF46B",
    "server": "uvicorn"
  },
  "status": "200 OK"
}

```

Copy prediction output into *argmax* to retrieve result of the image prediction:

```

>>> print(np.argmax([[0.0159152802079916,
>>>                    -1.5496551990509033,
>>>                    .....
>>>                    2.5116958618164062]]))
258

```

### 12.5.2.9 Example

```

from ads.common.model_metadata import UseCaseType
from ads.model.framework.pytorch_model import PyTorchModel

import numpy as np
from PIL import Image

import tempfile
import torchvision
from torchvision import transforms

import urllib

# Load a pretrained PyTorch Model
model = torchvision.models.resnet18(pretrained=True)
model.eval()

# Prepare Model Artifact for PyTorch Model
artifact_dir = tempfile.mkdtemp()
pytorch_model = PyTorchModel(model, artifact_dir=artifact_dir)
pytorch_model.prepare(
    inference_conda_env="pytorch110_p38_cpu_v1",
    training_conda_env="pytorch110_p38_cpu_v1",
    use_case_type=UseCaseType.IMAGE_CLASSIFICATION,
    force_overwrite=True,
    use_torch_script=True
)

# Download an image for running inference
url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog.jpg")
urllib.request.urlretrieve(url, filename)

```

(continues on next page)

(continued from previous page)

```

# Load image
input_image = Image.open(filename)
preprocess = transforms.Compose(
    [
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ]
)
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by the model

# Check if the artifacts are generated correctly.
# The verify method invokes the ``predict`` function defined inside ``score.py`` in the
↳ artifact_dir
prediction = pytorch_model.verify(input_batch)["prediction"]
print(np.argmax(prediction))

# Register the model
model_id = pytorch_model.save(display_name="PyTorch Model")

# Deploy and create an endpoint for the PyTorch model
pytorch_model.deploy(
    display_name="PyTorch Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxxx",
)

# Generate prediction by invoking the deployed endpoint
prediction = pytorch_model.predict(input_batch)["prediction"]

print(np.argmax(prediction))

# To delete the deployed endpoint uncomment the line below
# pytorch_model.delete_deployment(wait_for_completion=True)

```

### 12.5.3 TensorFlowModel

See [API Documentation](#)

### 12.5.3.1 Overview

The `ads.model.framework.tensorflow_model.TensorFlowModel` class in ADS is designed to allow you to rapidly get a TensorFlow model into production. The `.prepare()` method creates the model artifacts that are needed to deploy a functioning model without you having to configure it or write code. However, you can customize the required `score.py` file.

The `.verify()` method simulates a model deployment by calling the `load_model()` and `predict()` methods in the `score.py` file. With the `.verify()` method, you can debug your `score.py` file without deploying any models. The `.save()` method deploys a model artifact to the model catalog. The `.deploy()` method deploys a model to a REST endpoint.

The following steps take your trained TensorFlow model and deploy it into production with a few lines of code.

#### Create a TensorFlow Model

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(trainx, trainy), (testx, testy) = mnist.load_data()
trainx, testx = trainx / 255.0, testx / 255.0

model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10),
    ])
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer="adam", loss=loss_fn, metrics=["accuracy"])
model.fit(trainx, trainy, epochs=1)
```

### 12.5.3.2 Prepare Model Artifact

```
from ads.common.model_metadata import UseCaseType
from ads.model.framework.tensorflow_model import TensorFlowModel
import tempfile

artifact_dir = tempfile.mkdtemp()
tensorflow_model = TensorFlowModel(estimator=model, artifact_dir=artifact_dir)
tensorflow_model.prepare(
    inference_conda_env="tensorflow28_p38_cpu_v1",
    training_conda_env="tensorflow28_p38_cpu_v1",
    X_sample=trainx,
    y_sample=trainy,
    use_case_type=UseCaseType.MULTINOMIAL_CLASSIFICATION,
)
```

Instantiate a `ads.model.framework.tensorflow_model.TensorFlowModel()` object with a TensorFlow model. Each instance accepts the following parameters:

- `artifact_dir`: str: Artifact directory to store the files needed for deployment.



- `auth`: (Dict, optional): Defaults to None. The default authentication is set using the `ads.set_auth` API. To override the default, use `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()` and create the appropriate authentication signer and the `**kwargs` required to instantiate the `IdentityClient` object.
- `estimator`: Callable: Any model object generated by the TensorFlow framework.
- `properties`: (`ModelProperties`, optional): Defaults to None. The `ModelProperties` object required to save and deploy a model.

The `properties` is an instance of the `ModelProperties` class and has the following predefined fields:

- `bucket_uri`: str
- `compartment_id`: str
- `deployment_access_log_id`: str
- `deployment_bandwidth_mbps`: int
- `deployment_instance_count`: int
- `deployment_instance_shape`: str
- `deployment_log_group_id`: str
- `deployment_predict_log_id`: str
- `deployment_memory_in_gbs`: Union[float, int]
- `deployment_ocpus`: Union[float, int]
- `inference_conda_env`: str
- `inference_python_version`: str
- `overwrite_existing_artifact`: bool
- `project_id`: str
- `remove_existing_artifact`: bool
- `training_conda_env`: str
- `training_id`: str
- `training_python_version`: str
- `training_resource_id`: str
- `training_script_path`: str

By default, `properties` is populated from the environment variables when not specified. For example, in notebook sessions the environment variables are preset and stored in project id (`PROJECT_OCID`) and compartment id (`NB_SESSION_COMPARTMENT_OCID`). So `properties` populates these environment variables, and uses the values in methods such as `.save()` and `.deploy()`. Pass in values to overwrite the defaults. When you use a method that includes an instance of `properties`, then `properties` records the values that you pass in. For example, when you pass `inference_conda_env` into the `.prepare()` method, then `properties` records the value. To reuse the `properties` file in different places, you can export the `properties` file using the `.to_yaml()` method then reload it into a different machine using the `.from_yaml()` method.

### 12.5.3.3 Summary Status

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's Step column displays a Status of Done for the initiate step. And the Details column explains what the initiate step did such as generating a `score.py` file. The Step column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The Status column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

		Actions Needed
Step	Status	Details
initiate	Done	Initiated the model
prepare()	Available	Generated runtime.yaml
		Generated score.py
		Serialized model
		Populated metadata(Custom, Taxonomy and Provenance)
verify()	Not Available	Local tested .predict from score.py
save()	Not Available	Conducted Introspect Test
		Uploaded artifact to model catalog
deploy()	Not Available	Deployed the model
predict()	Not Available	Called deployment predict endpoint

In `TensorFlowModel`, data serialization is supported for JSON serializable objects. Plus, there is support for a dictionary, string, list, `np.ndarray`, and `tf.python.framework.ops.EagerTensor`. Not all these objects are JSON serializable, however, support to automatically serializes and deserializes is provided.

### 12.5.3.4 Register Model

```
>>> # Register the model
>>> model_id = tensorflow_model.save()
```

```
Start loading model.h5 from model directory /tmp/tmpapjjzeol ...
Model is successfully loaded.
['runtime.yaml', 'model.h5', 'score.py']

'ocid1.datasciencemodel.oc1.xxx.xxxxx'
```

### 12.5.3.5 Deploy and Generate Endpoint

```
>>> # Deploy and create an endpoint for the TensorFlow model
>>> tensorflow_model.deploy(
    display_name="TensorFlow Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)

>>> print(f"Endpoint: {tensorflow_model.model_deployment.url}")

https://modeldeployment.{region}.oci.customer-oci.com/ocid1.datasciencemodeldeployment.
↳ oc1.xxx.xxxxx
```

### 12.5.3.6 Run Prediction against Endpoint

```
# Generate prediction by invoking the deployed endpoint
tensorflow_model.predict(testx[:3])['prediction']
```

```
[[-2.9461750984191895, -5.293642997741699, 0.4030594229698181, 3.0270071029663086, -6.
↳ 470805644989014, -2.07453989982605, -9.646402359008789, 9.256569862365723, -2.
↳ 6433541774749756, -0.8167083263397217],
[-3.4297854900360107, 2.4863781929016113, 8.968724250793457, 3.162344217300415, -11.
↳ 153030395507812, 0.15335027873516083, -0.5451826453208923, -7.817524433135986, -1.
↳ 0585914850234985, -10.736929893493652],
[-4.420501232147217, 5.841022491455078, -0.17066864669322968, -1.0071465969085693, -2.
↳ 261953592300415, -3.0983355045318604, -2.0874621868133545, 1.0745809078216553, -1.
↳ 2511857748031616, -2.273810625076294]]
```

#### 12.5.3.6.1 Predict with Image

New in version 2.6.7.

Predict Image by passing a uri, which can be http(s), local path, or other URLs (e.g. starting with “oci://”, “s3://”, and “gcs://”), of the image or a PIL.Image.Image object using the `image` argument in `predict()` to predict a single image. The image will be converted to a tensor and then serialized so it can be passed to the endpoint. You can catch the tensor in `score.py` to perform further transformation.

Common example for model deployment prediction with image passed:

```
# Generate prediction by invoking the deployed endpoint
prediction = tensorflow_model.predict(image=<uri>)['prediction']
```

See the “Predict with Image” example [here](#).

### 12.5.3.7 Run Prediction with oci raw-request command

Model deployment endpoints can be invoked with the OCI-CLI. This example invokes a model deployment with the CLI with a `numpy.ndarray` payload:

#### 12.5.3.7.1 `numpy.ndarray` payload example

```
>>> # Prepare data sample for prediction and save it to file 'data-payload'
>>> from io import BytesIO
>>> import base64
>>> import numpy as np

>>> data = testx[:3]
>>> np_bytes = BytesIO()
>>> np.save(np_bytes, data, allow_pickle=True)
>>> data = base64.b64encode(np_bytes.getvalue()).decode("utf-8")
>>> with open('data-payload', 'w') as f:
>>>     f.write('{"data": "' + data + '", "data_type": "numpy.ndarray"}')
```

File `data-payload` will have this information:

```
{"data": "k05VTVBZAQB2AHsnZGVzY3InOiAnPGY4JywgJ2ZvcnRyYW5fb3JkZXInOi...
.....AAAAAAAAAAAAAAAAAAAAAA=", "data_type": "numpy.ndarray"}
```

Use file `data-payload` with data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↪datasciencemodeldeployment.oc1.xxx.xxxx/predict
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body file://data-payload
```

#### 12.5.3.7.2 Expected output of raw-request command

```
{
  "data": {
    "prediction": [
      [
        -1.8818638324737549,
        -6.04175329208374,
        .....,
        -1.1257498264312744
      ],
      [
        1.2170600891113281,
        1.6379727125167847,
        .....,
        -9.877771377563477
      ]
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```

    ],
    [
        -4.255424499511719,
        5.320354461669922,
        .....,
        -2.85855555343628
    ]
]
},
"headers": {
    "Connection": "keep-alive",
    "Content-Length": "594",
    "Content-Type": "application/json",
    "Date": "Thu, 08 Dec 2022 23:25:47 GMT",
    "X-Content-Type-Options": "nosniff",
    "opc-request-id": "E70002DAA3024F46B074F9B53DB6BEBB/421B34FCB12CF33F23C85D5619A62926/
↪CAABF4A269C63B112482B2E57463CA13",
    "server": "uvicorn"
},
"status": "200 OK"
}

```

### 12.5.3.8 Example

```

from ads.common.model_metadata import UseCaseType
from ads.model.framework.tensorflow_model import TensorFlowModel

import tensorflow as tf

import tempfile

# Load MNIST Data
mnist = tf.keras.datasets.mnist
(trainx, trainy), (testx, testy) = mnist.load_data()
trainx, testx = trainx / 255.0, testx / 255.0

# Train TensorFlow model
model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10),
    ]
)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer="adam", loss=loss_fn, metrics=["accuracy"])
model.fit(trainx, trainy, epochs=1)

```

(continues on next page)

(continued from previous page)

```

artifact_dir = tempfile.mkdtemp()

# Prepare Model Artifact for TensorFlow model
tensorflow_model = TensorFlowModel(estimator=model, artifact_dir=artifact_dir)
tensorflow_model.prepare(
    inference_conda_env="tensorflow28_p38_cpu_v1",
    training_conda_env="tensorflow28_p38_cpu_v1",
    X_sample=trainx,
    y_sample=trainy,
    use_case_type=UseCaseType.MULTINOMIAL_CLASSIFICATION,
)

# Check if the artifacts are generated correctly.
# The verify method invokes the ``predict`` function defined inside ``score.py`` in the
↪ artifact_dir
tensorflow_model.verify(testx[:10])["prediction"]

# Register the model
model_id = tensorflow_model.save(display_name="TensorFlow Model")

# Deploy and create an endpoint for the TensorFlow model
tensorflow_model.deploy(
    display_name="TensorFlow Model For Classification",
    deployment_log_group_id="ocidl.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocidl.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocidl.log.oc1.xxx.xxxxx",
)

# Generate prediction by invoking the deployed endpoint
tensorflow_model.predict(testx)["prediction"]

# To delete the deployed endpoint uncomment the line below
# tensorflow_model.delete_deployment(wait_for_completion=True)

```

## 12.5.4 SparkPipelineModel

See [API Documentation](#)

### 12.5.4.1 Overview

The `SparkPipelineModel` class in ADS is designed to allow you to rapidly get a PySpark model into production. The `.prepare()` method creates the model artifacts that are needed to deploy a functioning model without you having to configure it or write code. However, you can customize the required `score.py` file.

The `.verify()` method simulates a model deployment by calling the `load_model()` and `predict()` methods in the `score.py` file. With the `.verify()` method, you can debug your `score.py` file without deploying any models. The `.save()` method deploys a model artifact to the model catalog. The `.deploy()` method deploys a model to a REST endpoint.

The following steps take your trained PySpark model and deploy it into production with a few lines of code.

#### Create a Spark Pipeline Model

Generate a synthetic dataset:

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .getOrCreate()
training = spark.createDataFrame(
    [
        (0, "a b c d e spark", 1.0),
        (1, "b d", 0.0),
        (2, "spark f g h", 1.0),
        (3, "hadoop mapreduce", 0.0),
    ],
    ["id", "text", "label"],
)
test = spark.createDataFrame(
    [
        (4, "spark i j k"),
        (5, "l m n"),
        (6, "spark hadoop spark"),
        (7, "apache hadoop"),
    ],
    ["id", "text"],
)
```

Create a Spark Pipeline. (Note that a Spark Pipeline can be made with just 1 stage.)

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)

pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
model = pipeline.fit(training)
```

#### 12.5.4.2 Prepare Model Artifact

```
import tempfile
from ads.model.framework.spark_model import SparkPipelineModel
from ads.common.model_metadata import UseCaseType

artifact_dir=tempfile.mkdtemp()
spark_model = SparkPipelineModel(estimator=model, artifact_dir=artifact_dir)

spark_model.prepare(inference_conda_env="pyspark32_p38_cpu_v2",
                    X_sample=training,
```

(continues on next page)

(continued from previous page)

```
force_overwrite=True,
use_case_type=UseCaseType.BINARY_CLASSIFICATION)
```

Instantiate a `SparkPipelineModel()` object with a PySpark model. Each instance accepts the following parameters:

- `artifact_dir`: str. Artifact directory to store the files needed for deployment.
- `auth`: (Dict, optional): Defaults to `None`. The default authentication is set using the `ads.set_auth` API. To override the default, use `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()` and create the appropriate authentication signer and the `**kwargs` required to instantiate the `IdentityClient` object.
- `estimator`: Callable. Any model object generated by the PySpark framework.
- `properties`: (`ModelProperties`, optional). Defaults to `None`. The `ModelProperties` object required to save and deploy model.

The `properties` is an instance of the `ModelProperties` class and has the following predefined fields:

- `bucket_uri`: str
- `compartment_id`: str
- `deployment_access_log_id`: str
- `deployment_bandwidth_mbps`: int
- `deployment_instance_count`: int
- `deployment_instance_shape`: str
- `deployment_log_group_id`: str
- `deployment_predict_log_id`: str
- `deployment_memory_in_gbs`: Union[float, int]
- `deployment_ocpus`: Union[float, int]
- `inference_conda_env`: str
- `inference_python_version`: str
- `overwrite_existing_artifact`: bool
- `project_id`: str
- `remove_existing_artifact`: bool
- `training_conda_env`: str
- `training_id`: str
- `training_python_version`: str
- `training_resource_id`: str
- `training_script_path`: str

By default, `properties` is populated from the environment variables when not specified. For example, in notebook sessions the environment variables are preset and stored in project id (`PROJECT_OCID`) and compartment id (`NB_SESSION_COMPARTMENT_OCID`). So `properties` populates these environment variables, and uses the values in methods such as `.save()` and `.deploy()`. Pass in values to overwrite the defaults. When you use a method that includes an instance of `properties`, then `properties` records the values that you pass in. For example, when you pass `inference_conda_env` into the `.prepare()` method, then `properties` records the value. To reuse the `properties`



file in different places, you can export the properties file using the `.to_yaml()` method then reload it into a different machine using the `.from_yaml()` method.

### 12.5.4.3 Summary Status

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's Step column displays a Status of Done for the initiate step. And the Details column explains what the initiate step did such as generating a `score.py` file. The Step column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The Status column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

		Actions Needed
Step	Status	Details
initiate	Done	Initiated the model
prepare()	Available	Generated runtime.yaml
		Generated score.py
		Serialized model
		Populated metadata(Custom, Taxonomy and Provenance)
verify()	Not Available	Local tested .predict from score.py
save()	Not Available	Conducted Introspect Test
		Uploaded artifact to model catalog
deploy()	Not Available	Deployed the model
predict()	Not Available	Called deployment predict endpoint

### 12.5.4.4 Register Model

```
model_id = spark_model.save()
```

```
Start loading model.joblib from model directory /tmp/tmpphdo8dfn3 ...
```

```
Model is successfully loaded.
```

```
['input_schema.json', 'runtime.yaml', 'model_input_data_schema.json', 'model', 'score.py  
→']
```

```
'ocid1.datasciencemodel.oc1.xxx.xxxxx'
```

### 12.5.4.5 Deploy and Generate Endpoint

```
spark_model.deploy(  
    display_name="Spark Pipeline Model For Classification",  
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",  
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",  
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",  
)  
  
print(f"Endpoint: {spark_model.model_deployment.url}")  
  
# https://modeldeployment.{region}.oci.customer-oci.com/ocid1.datasciencemodeldeployment.  
↪ oc1.xxx.xxxxx
```

### 12.5.4.6 Run Prediction against Endpoint

```
spark_model.predict(test)['prediction']  
# [0.0, 0.0, 1.0, 0.0]
```

### 12.5.4.7 Run Prediction with oci raw-request command

Model deployment endpoints can be invoked with the OCI-CLI. This example invokes a model deployment with the CLI with a json payload:

#### 12.5.4.7.1 json payload example

```
>>> # Prepare data sample for prediction and save it to file 'payload'  
>>> print(json.dumps(test.toJSON().collect()))  
[{"id":4,"text":"spark i j k"}, {"id":5,"text":"l m n"},  
{"id":6,"text":"spark hadoop spark"}, {"id":7,"text":"apache hadoop"}]
```

Use printed output of the data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.  
↪ datasciencemodeldeployment.oc1.xxx.xxxxx/predict  
export data='{"data": [{"id":4,"text":"spark i j k"}, ... {"id":7,"text":"  
↪ apache hadoop"}]}'  
oci raw-request \  
    --http-method POST \  
    --target-uri $uri \  
    --request-body "$data"
```

### 12.5.4.7.2 Expected output of raw-request command

```
{
  "data": {
    "prediction": [
      0.0,
      0.0,
      1.0,
      0.0
    ]
  },
  "headers": {
    "Connection": "keep-alive",
    "Content-Length": "32",
    "Content-Type": "application/json",
    "Date": "Thu, 08 Dec 2022 18:45:12 GMT",
    "X-Content-Type-Options": "nosniff",
    "opc-request-id": "C2E73B1679B34BAD8358B49D20619055/0EE2E5F93F48142725525D7A5BA7F5FB/049A66AA38AA0163DBBC70F225285851",
    "server": "uvicorn"
  },
  "status": "200 OK"
}
```

### 12.5.4.8 Example

Adapted from an example provided by Apache in the PySpark API Reference Documentation.

```
import tempfile
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import SparkSession
from ads.model.framework.spark_model import SparkPipelineModel
from ads.common.model_metadata import UseCaseType

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .getOrCreate()

artifact_dir=tempfile.mkdtemp()

training = spark.createDataFrame(
    [
        (0, "a b c d e spark", 1.0),
        (1, "b d", 0.0),
        (2, "spark f g h", 1.0),
        (3, "hadoop mapreduce", 0.0),
    ],
    ["id", "text", "label"],
)
```

(continues on next page)

(continued from previous page)

```

test = spark.createDataFrame(
    [
        (4, "spark i j k"),
        (5, "l m n"),
        (6, "spark hadoop spark"),
        (7, "apache hadoop"),
    ],
    ["id", "text"],
)

tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

model = pipeline.fit(training)

spark_model = SparkPipelineModel(estimator=model, artifact_dir=artifact_dir)

spark_model.prepare(inference_conda_env="pyspark32_p38_cpu_v2",
                    X_sample=training,
                    force_overwrite=True,
                    use_case_type=UseCaseType.BINARY_CLASSIFICATION)

# Check if the artifacts are generated correctly.
# The verify method invokes the ``predict`` function defined inside ``score.py`` in the
↪ artifact_dir
prediction = spark_model.verify(test)

# Register the model
spark_model.save(display_name="Spark Pipeline Model")

# Deploy and create an endpoint for the Spark model
spark_model.deploy(
    display_name="Spark Pipeline Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)

# Generate prediction by invoking the deployed endpoint
spark_model.predict(test)["prediction"]

# To delete the deployed endpoint uncomment the line below
# spark_model.delete_deployment(wait_for_completion=True)

```

## 12.5.5 LightGBMModel

See [API Documentation](#)

### 12.5.5.1 Overview

The `ads.model.framework.lightgbm_model.LightGBMModel` class in ADS is designed to allow you to rapidly get a LightGBM model into production. The `.prepare()` method creates the model artifacts that are needed to deploy a functioning model without you having to configure it or write code. However, you can customize the required `score.py` file.

The `.verify()` method simulates a model deployment by calling the `load_model()` and `predict()` methods in the `score.py` file. With the `.verify()` method, you can debug your `score.py` file without deploying any models. The `.save()` method deploys a model artifact to the model catalog. The `.deploy()` method deploys a model to a REST endpoint.

The following steps take your trained LightGBM model and deploy it into production with a few lines of code.

The `LightGBMModel` module in ADS supports serialization for models generated from both the [Training API](#) using `lightgbm.train()` and the [Scikit-Learn API](#) using `lightgbm.LGBMClassifier()`. Both of these interfaces are defined by `LightGBM`.

The Training API in `LightGBM` contains training and cross-validation routines. The `Dataset` class is an internal data structure that is used by `LightGBM` when using the `lightgbm.train()` method. You can also create `LightGBM` models using the Scikit-Learn Wrapper interface. The `LightGBMModel` class handles the differences between the `LightGBM` Training and SciKit-Learn APIs seamlessly.

#### Create LightGBM Model

```
import lightgbm as lgb
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

seed = 42

X, y = make_classification(n_samples=10000, n_features=15, n_classes=2, flip_y=0.05)
trainx, testx, trainy, testy = train_test_split(X, y, test_size=30, random_state=seed)
model = lgb.LGBMClassifier(
    n_estimators=100, learning_rate=0.01, random_state=42
)
model.fit(
    trainx,
    trainy,
)
```

### 12.5.5.2 Prepare Model Artifact

```
from ads.common.model_metadata import UseCaseType
from ads.model.framework.lightgbm_model import LightGBMModel

artifact_dir = tempfile.mkdtemp()
lightgbm_model = LightGBMModel(estimator=model, artifact_dir=artifact_dir)
lightgbm_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
```

(continues on next page)

(continued from previous page)

```
training_conda_env="generalml_p38_cpu_v1",
X_sample=trainx,
y_sample=trainy,
use_case_type=UseCaseType.BINARY_CLASSIFICATION,
)
```

Instantiate a `ads.model.framework.lightgbm_model.LightGBMModel()` object with a LightGBM model. Each instance accepts the following parameters:

- `artifact_dir: str`: Artifact directory to store the files needed for deployment.
- `auth: (Dict, optional)`: Defaults to `None`. The default authentication is set using the `ads.set_auth` API. To override the default, use `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()` and create the appropriate authentication signer and the `**kwargs` required to instantiate the `IdentityClient` object.
- `estimator: (Callable)`: Trained LightGBM model using the Training API or the Scikit-Learn Wrapper interface.
- `properties: (ModelProperties, optional)`: Defaults to `None`. The `ModelProperties` object required to save and deploy a model.

The `properties` is an instance of the `ModelProperties` class and has the following predefined fields:

- `bucket_uri: str`
- `compartment_id: str`
- `deployment_access_log_id: str`
- `deployment_bandwidth_mbps: int`
- `deployment_instance_count: int`
- `deployment_instance_shape: str`
- `deployment_log_group_id: str`
- `deployment_predict_log_id: str`
- `deployment_memory_in_gbs: Union[float, int]`
- `deployment_ocpus: Union[float, int]`
- `inference_conda_env: str`
- `inference_python_version: str`
- `overwrite_existing_artifact: bool`
- `project_id: str`
- `remove_existing_artifact: bool`
- `training_conda_env: str`
- `training_id: str`
- `training_python_version: str`
- `training_resource_id: str`
- `training_script_path: str`

By default, `properties` is populated from the environment variables when not specified. For example, in notebook sessions the environment variables are preset and stored in project id (`PROJECT_OCID`) and compartment id (`NB_SESSION_COMPARTMENT_OCID`). So `properties` populates these environment variables, and uses the values in methods such as `.save()` and `.deploy()`. Pass in values to overwrite the defaults. When you use a method that includes an instance of `properties`, then `properties` records the values that you pass in. For example, when you pass `inference_conda_env` into the `.prepare()` method, then `properties` records the value. To reuse the `properties` file in different places, you can export the `properties` file using the `.to_yaml()` method then reload it into a different machine using the `.from_yaml()` method.

### 12.5.5.3 Summary Status

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's Step column displays a Status of Done for the initiate step. And the Details column explains what the initiate step did such as generating a `score.py` file. The Step column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The Status column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

		Actions Needed
Step	Status	Details
initiate	Done	Initiated the model
prepare()	Available	Generated runtime.yaml
		Generated score.py
		Serialized model
		Populated metadata(Custom, Taxonomy and Provenance)
verify()	Not Available	Local tested .predict from score.py
save()	Not Available	Conducted Introspect Test
		Uploaded artifact to model catalog
deploy()	Not Available	Deployed the model
predict()	Not Available	Called deployment predict endpoint

#### 12.5.5.4 Register Model

```
>>> # Register the model
>>> model_id = lightgbm_model.save()

Start loading model.joblib from model directory /tmp/tmp9l0uhtbb ...
Model is successfully loaded.
['runtime.yaml', 'model.joblib', 'score.py', 'input_schema.json']

'ocid1.datasciencemodel.oc1.xxx.xxxxx'
```

#### 12.5.5.5 Deploy and Generate Endpoint

```
>>> # Deploy and create an endpoint for the LightGBM model
>>> lightgbm_model.deploy(
    display_name="LightGBM Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)

>>> print(f"Endpoint: {lightgbm_model.model_deployment.url}")

https://modeldeployment.{region}.oci.customer-oci.com/ocid1.datasciencemodeldeployment.
  ↪ oc1.xxx.xxxxx
```

#### 12.5.5.6 Run Prediction against Endpoint

```
# Generate prediction by invoking the deployed endpoint
lightgbm_model.predict(testx)['prediction']
```

```
[1,0,...,1]
```



### 12.5.5.7 Run Prediction with oci raw-request command

Model deployment endpoints can be invoked with the OCI-CLI. The below examples invoke a model deployment with the CLI with different types of payload: `json`, `numpy.ndarray`, `pandas.core.frame.DataFrame` or `dict`.

#### 12.5.5.7.1 *json* payload example

```
>>> # Prepare data sample for prediction
>>> data = testx[[11]]
>>> data
array([[ 0.59990614,  0.95516275, -1.22366985, -0.89270887, -1.14868768,
        -0.3506047 ,  0.28529227,  2.00085413,  0.31212668,  0.39411511,
         0.87301082, -0.01743923, -1.15089633,  1.03461823,  1.50228029]])
```

Use printed output of the data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↳datasciencemodeldeployment.oc1.xxx.xxxxx/predict
export data='{"data": [[ 0.59990614,  0.95516275, ... , 1.50228029]]}'
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"
```

#### 12.5.5.7.2 *numpy.ndarray* payload example

```
>>> # Prepare data sample for prediction
>>> from io import BytesIO
>>> import base64
>>> import numpy as np

>>> data = testx[[10]]
>>> np_bytes = BytesIO()
>>> np.save(np_bytes, data, allow_pickle=True)
>>> data = base64.b64encode(np_bytes.getvalue()).decode("utf-8")
>>> print(data)
k05VTVBZAQB2AHsnZGVzY.....D1cJ+D8=
```

Use printed output of *base64* data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↳datasciencemodeldeployment.oc1.xxx.xxxxx/predict
export data='{"data": "k05VTVBZAQB2AHsnZGVzY.....4UdN0L8=", "data_type": "numpy.ndarray"}'
↳
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"
```

### 12.5.5.7.3 *pandas.core.frame.DataFrame* payload example

```
>>> # Prepare data sample for prediction
>>> import pandas as pd

>>> df = pd.DataFrame(testx[[10]])
>>> print(json.dumps(df.to_json()))
{"0":{"0":{"0":0.4133005141},"1":{"0":{"0":0.676589266},...,"14":{"0":{"0":-0.2547168443}}}"
```

Use printed output of DataFrame data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↪datasciencemodeldeployment.oc1.xxx.xxxx/predict
export data='{"data":{"0":{"0":{"0":0.4133005141},...,"14":{"0":{"0":-0.2547168443}}}',
↪"data_type":"pandas.core.frame.DataFrame"}'
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"
```

### 12.5.5.7.4 *dict* payload example

```
>>> # Prepare data sample for prediction
>>> import pandas as pd
```

```
>>> df = pd.DataFrame(testx[[11]])
>>> print(json.dumps(df.to_dict()))
{"0": {"0": 0.5999061426438217}, "1": {"0": 0.9551627492226553}, ..., "14": {"0": 1.
↪5022802918908846}}
```

Use printed output of dict data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↪datasciencemodeldeployment.oc1.xxx.xxxx/predict
export data='{"data": {"0": {"0": 0.5999061426438217}, ..., "14": {"0": 1.
↪5022802918908846}}}'
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"
```

### 12.5.5.7.5 Expected output of raw-request command

```
{
  "data": {
    "prediction": [
      1
    ]
  },
}
```

(continues on next page)

(continued from previous page)

```

"headers": {
  "Connection": "keep-alive",
  "Content-Length": "18",
  "Content-Type": "application/json",
  "Date": "Wed, 07 Dec 2022 18:31:05 GMT",
  "X-Content-Type-Options": "nosniff",
  "opc-request-id": "B67EED723ADD43D7BBA1AD5AFCCBD0C6/03F218FF833BC8A2D6A5BDE4AB8B7C12/
↪6ACBC4E5C5127AC80DA590568D628B60",
  "server": "uvicorn"
},
"status": "200 OK"
}

```

### 12.5.5.8 Example

```

from ads.model.framework.lightgbm_model import LightGBMModel
from ads.common.model_metadata import UseCaseType

import lightgbm as lgb

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

import tempfile

seed = 42

# Create a classification dataset
X, y = make_classification(n_samples=10000, n_features=15, n_classes=2, flip_y=0.05)

trainx, testx, trainy, testy = train_test_split(X, y, test_size=30, random_state=seed)

# Train LGBM model
model = lgb.LGBMClassifier(n_estimators=100, learning_rate=0.01, random_state=42)
model.fit(
    trainx,
    trainy,
)

# Prepare Model Artifact for LightGBM model
artifact_dir = tempfile.mkdtemp()
lightgbm_model = LightGBMModel(estimator=model, artifact_dir=artifact_dir)
lightgbm_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    training_conda_env="generalml_p38_cpu_v1",
    X_sample=trainx,
    y_sample=trainy,
    force_overwrite=True,
    use_case_type=UseCaseType.BINARY_CLASSIFICATION,
)

```

(continues on next page)

(continued from previous page)

```

# Check if the artifacts are generated correctly.
# The verify method invokes the ``predict`` function defined inside ``score.py`` in the
↳ artifact_dir
lightgbm_model.verify(testx[:10])["prediction"]

# Register the model
model_id = lightgbm_model.save(display_name="LightGBM Model")

# Deploy and create an endpoint for the LightGBM model
lightgbm_model.deploy(
    display_name="LightGBM Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)

print(f"Endpoint: {lightgbm_model.model_deployment.url}")

# Generate prediction by invoking the deployed endpoint
lightgbm_model.predict(testx)["prediction"]

# To delete the deployed endpoint uncomment the line below
# lightgbm_model.delete_deployment(wait_for_completion=True)

```

## 12.5.6 XGBoostModel

See [API Documentation](#)

### 12.5.6.1 Overview

The `ads.model.framework.xgboost_model.XGBoostModel` class in ADS is designed to allow you to rapidly get a XGBoost model into production. The `.prepare()` method creates the model artifacts that are needed to deploy a functioning model without you having to configure it or write code. However, you can customize the required `score.py` file.

The `.verify()` method simulates a model deployment by calling the `load_model()` and `predict()` methods in the `score.py` file. With the `.verify()` method, you can debug your `score.py` file without deploying any models. The `.save()` method deploys a model artifact to the model catalog. The `.deploy()` method deploys a model to a REST endpoint.

The following steps take your trained XGBoost model and deploy it into production with a few lines of code.

The XGBoostModel module in ADS supports serialization for models generated from both the [Learning API](#) using `xgboost.train()` and the [Scikit-Learn API](#) using `xgboost.XGBClassifier()`. Both of these interfaces are defined by XGBoost.

#### Create XGBoost Model

```

from ads.model.framework.xgboost_model import XGBoostModel
from ads.common.model_metadata import UseCaseType

```

(continues on next page)

(continued from previous page)

```

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

import tempfile

import xgboost

seed = 42

X, y = make_classification(n_samples=10000, n_features=15, n_classes=2, flip_y=0.05)
trainx, testx, trainy, testy = train_test_split(X, y, test_size=30, random_state=seed)
model = xgboost.XGBClassifier(
    n_estimators=100, learning_rate=0.01, random_state=42, use_label_encoder=False
)
model.fit(
    trainx,
    trainy,
)

```

### 12.5.6.2 Prepare Model Artifact

```

from ads.model.framework.xgboost_model import XGBoostModel
from ads.common.model_metadata import UseCaseType

artifact_dir = tempfile.mkdtemp()
xgb_model = XGBoostModel(estimator=model, artifact_dir=artifact_dir)
xgb_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    training_conda_env="generalml_p38_cpu_v1",
    X_sample=trainx,
    y_sample=trainy,
    use_case_type=UseCaseType.BINARY_CLASSIFICATION,
)

```

Instantiate a `ads.model.framework.xgboost_model.XGBoostModel` object with an XGBoost model. Each instance accepts the following parameters:

- `artifact_dir`: str: Artifact directory to store the files needed for deployment.
- `auth`: (Dict, optional): Defaults to None. The default authentication is set using the `ads.set_auth` API. To override the default, use `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()` and create the appropriate authentication signer and the `**kwargs` required to instantiate the `IdentityClient` object.
- `estimator`: (Callable): Trained XGBoost model either using the Learning API or the Scikit-Learn Wrapper interface.
- `properties`: (ModelProperties, optional): Defaults to None. The `ModelProperties` object required to save and deploy a model.

The `properties` is an instance of the `ModelProperties` class and has the following predefined fields:

- `bucket_uri`: str

- `compartment_id`: str
- `deployment_access_log_id`: str
- `deployment_bandwidth_mbps`: int
- `deployment_instance_count`: int
- `deployment_instance_shape`: str
- `deployment_log_group_id`: str
- `deployment_predict_log_id`: str
- `deployment_memory_in_gbs`: Union[float, int]
- `deployment_ocpus`: Union[float, int]
- `inference_conda_env`: str
- `inference_python_version`: str
- `overwrite_existing_artifact`: bool
- `project_id`: str
- `remove_existing_artifact`: bool
- `training_conda_env`: str
- `training_id`: str
- `training_python_version`: str
- `training_resource_id`: str
- `training_script_path`: str

By default, `properties` is populated from the environment variables when not specified. For example, in notebook sessions the environment variables are preset and stored in project id (`PROJECT_OCID`) and compartment id (`NB_SESSION_COMPARTMENT_OCID`). So `properties` populates these environment variables, and uses the values in methods such as `.save()` and `.deploy()`. Pass in values to overwrite the defaults. When you use a method that includes an instance of `properties`, then `properties` records the values that you pass in. For example, when you pass `inference_conda_env` into the `.prepare()` method, then `properties` records the value. To reuse the `properties` file in different places, you can export the `properties` file using the `.to_yaml()` method then reload it into a different machine using the `.from_yaml()` method.

### 12.5.6.3 Summary Status

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's `Step` column displays a Status of Done for the initiate step. And the `Details` column explains what the initiate step did such as generating a `score.py` file. The `Step` column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The `Status` column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

		Actions Needed
Step	Status	Details
initiate	Done	Initiated the model
prepare()	Available	Generated runtime.yaml
		Generated score.py
		Serialized model
		Populated metadata(Custom, Taxonomy and Provenance)
verify()	Not Available	Local tested .predict from score.py
save()	Not Available	Conducted Introspect Test
		Uploaded artifact to model catalog
deploy()	Not Available	Deployed the model
predict()	Not Available	Called deployment predict endpoint

#### 12.5.6.4 Register Model

```
>>> # Register the model
>>> model_id = xgb_model.save()

Start loading model.joblib from model directory /tmp/tmp9l0uhtbb ...
Model is successfully loaded.
['runtime.yaml', 'model.joblib', 'score.py', 'input_schema.json']

'ocid1.datasciencemodel.oc1.xxx.xxxxx'
```

#### 12.5.6.5 Deploy and Generate Endpoint

```
>>> # Deploy and create an endpoint for the XGBoost model
>>> xgb_model.deploy(
    display_name="XGBoost Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)

>>> print(f"Endpoint: {xgb_model.model_deployment.url}")

https://modeldeployment.{region}.oci.customer-oci.com/ocid1.datasciencemodeldeployment.
↳ oc1.xxx.xxxxx
```

### 12.5.6.6 Run Prediction against Endpoint

```
# Generate prediction by invoking the deployed endpoint
>>> xgb_model.predict(testx)['prediction']
[0.22879330813884735, 0.2054443359375, 0.20657016336917877, ..., 0.8005291223526001]
```

### 12.5.6.7 Run Prediction with oci raw-request command

Model deployment endpoints can be invoked with the OCI-CLI. The below examples invoke a model deployment with the CLI with different types of payload: `json`, `numpy.ndarray`, `pandas.core.frame.DataFrame` or `dict`.

#### 12.5.6.7.1 json payload example

```
>>> # Prepare data sample for prediction
>>> data = testx[[12]]
>>> data
array([[ 0.66098176, -1.06487896, -0.88581208,  0.05667259,  0.42884393,
        -0.52552184,  0.75322749, -0.58112776, -0.81102029, -1.35854886,
        -1.16440502, -0.67791303, -0.04810906,  0.72970972, -0.24120756]])
```

Use printed output of the data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↳datasciencemodeldeployment.oc1.xxx.xxxxx/predict
export data='{"data": [[ 0.66098176, -1.06487896, ... , -0.24120756]]}'
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"
```

#### 12.5.6.7.2 numpy.ndarray payload example

```
>>> # Prepare data sample for prediction
>>> from io import BytesIO
>>> import base64
>>> import numpy as np

>>> data = testx[[12]]
>>> np_bytes = BytesIO()
>>> np.save(np_bytes, data, allow_pickle=True)
>>> data = base64.b64encode(np_bytes.getvalue()).decode("utf-8")
>>> print(data)
k05VTVBZAQB2AHsnZGVzY.....pePfzr8=
```

Use printed output of base64 data and endpoint to invoke prediction with raw-request command in terminal:

```
export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
↳datasciencemodeldeployment.oc1.xxx.xxxxx/predict
export data='{"data": "k05VTVBZAQB2AHsnZGVzY.....pePfzr8=", "data_type": "numpy.ndarray"}'
```

(continues on next page)



(continued from previous page)

```

❏ '
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"

```

### 12.5.6.7.3 *pandas.core.frame.DataFrame* payload example

```

>>> # Prepare data sample for prediction
>>> import pandas as pd

>>> df = pd.DataFrame(testx[[12]])
>>> print(json.dumps(df.to_json()))
{"0":{"0":{"0":0.6609817554},"1":{"0":-1.0648789569},...,"14":{"0":-0.2412075575}}
❏ }

```

Use printed output of *DataFrame* data and endpoint to invoke prediction with raw-request command in terminal:

```

export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
❏ datasciencemodeldeployment.oc1.xxx.xxxx/predict
export data='{"data":{"0":{"0":0.6609817554},...,"14":{"0":-0.2412075575}}',
❏ "data_type":"pandas.core.frame.DataFrame"}'
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"

```

### 12.5.6.7.4 *dict* payload example

```

>>> # Prepare data sample for prediction
>>> import pandas as pd

```

```

>>> df = pd.DataFrame(testx[[12]])
>>> print(json.dumps(df.to_dict()))
{"0": {"0": -0.6712208871908425}, "1": {"0": 0.5266565978285116}, ..., "14": {"0": 0.
❏ 9062102978188604}}

```

Use printed output of dict data and endpoint to invoke prediction with raw-request command in terminal:

```

export uri=https://modeldeployment.{region}.oci.customer-oci.com/ocid1.
❏ datasciencemodeldeployment.oc1.xxx.xxxx/predict
export data='{"data": {"0": {"0": -0.6712208871908425}, ..., "14": {"0": 0.
❏ 9062102978188604}}}'
oci raw-request \
  --http-method POST \
  --target-uri $uri \
  --request-body "$data"

```

## 12.5.6.7.5 Expected output of raw-request command

```
{
  "data": {
    "prediction": [
      0.5611757040023804
    ]
  },
  "headers": {
    "Connection": "keep-alive",
    "Content-Length": "35",
    "Content-Type": "application/json",
    "Date": "Wed, 07 Dec 2022 17:27:17 GMT",
    "X-Content-Type-Options": "nosniff",
    "opc-request-id": "19E90A7F2AFE401BB437DBC6168D2F1C/4A1CC9969F40B9F0656DD0497A28B51A/FEB42D1B690E8A665244046C7A151AB5",
    "server": "uvicorn"
  },
  "status": "200 OK"
}
```

## 12.5.6.8 Example

```
from ads.model.framework.xgboost_model import XGBoostModel
from ads.common.model_metadata import UseCaseType

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

import tempfile

import xgboost

seed = 42

# Create a classification dataset
X, y = make_classification(n_samples=10000, n_features=15, n_classes=2, flip_y=0.05)
trainx, testx, trainy, testy = train_test_split(X, y, test_size=30, random_state=seed)

# Train XGBoost model
model = xgboost.XGBClassifier(n_estimators=100, learning_rate=0.01, random_state=42)
model.fit(
    trainx,
    trainy,
)

artifact_dir = tempfile.mkdtemp()
xgb_model = XGBoostModel(estimator=model, artifact_dir=artifact_dir)
xgb_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    training_conda_env="generalml_p38_cpu_v1",
)
```

(continues on next page)

(continued from previous page)

```

X_sample=trainx,
y_sample=trainy,
use_case_type=UseCaseType.BINARY_CLASSIFICATION,
)

# Check if the artifacts are generated correctly.
# The verify method invokes the ``predict`` function defined inside ``score.py`` in the
↪ artifact_dir
xgb_model.verify(testx)

# Register the model
model_id = xgb_model.save(display_name="XGBoost Model")

# Deploy and create an endpoint for the XGBoost model
xgb_model.deploy(
    display_name="XGBoost Model For Classification",
    deployment_log_group_id="ocid1.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocid1.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocid1.log.oc1.xxx.xxxxx",
)

print(f"Endpoint: {xgb_model.model_deployment.url}")

# Generate prediction by invoking the deployed endpoint
xgb_model.predict(testx)["prediction"]

# To delete the deployed endpoint uncomment the line below
# xgb_model.delete_deployment(wait_for_completion=True)

```

## 12.5.7 AutoMLModel

See [API Documentation](#)

### 12.5.7.1 Overview

The `ads.model.framework.automl_model.AutoMLModel` class in ADS is designed to rapidly get your AutoML model into production. The `.prepare()` method creates the model artifacts needed to deploy the model without you having to configure it or write code. The `.prepare()` method serializes the model and generates a `runtime.yaml` and a `score.py` file that you can later customize.

The `.verify()` method simulates a model deployment by calling the `load_model()` and `predict()` methods in the `score.py` file. With the `.verify()` method, you can debug your `score.py` file without deploying any models. The `.save()` method deploys a model artifact to the model catalog. The `.deploy()` method deploys a model to a REST endpoint.

The following steps take your trained AutoML model and deploy it into production with a few lines of code.

#### Creating an Oracle Labs AutoML Model

Create an `OracleAutoMLProvider` object and use it to define how an Oracle Labs AutoML model is trained.

```
import logging
from ads.automl.driver import AutoML
from ads.automl.provider import OracleAutoMLProvider
from ads.dataset.dataset_browser import DatasetBrowser

ds = DatasetBrowser.sklearn().open("wine").set_target("target")
train, test = ds.train_test_split(test_size=0.1, random_state = 42)

ml_engine = OracleAutoMLProvider(n_jobs=-1, loglevel=logging.ERROR)
oracle_automl = AutoML(train, provider=ml_engine)
model, baseline = oracle_automl.train(
    model_list=['LogisticRegression', 'DecisionTreeClassifier'],
    random_state = 42, time_budget = 500)
```

### 12.5.7.2 Initialize

Instantiate an `AutoMLModel()` object with an AutoML model. Each instance accepts the following parameters:

- `artifact_dir`: str: Artifact directory to store the files needed for deployment.
- `auth`: (Dict, optional): Defaults to None. The default authentication is set using the `ads.set_auth` API. To override the default, use `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()` and create the appropriate authentication signer and the `**kwargs` required to instantiate the `IdentityClient` object.
- `estimator`: (Callable): Trained AutoML model.
- `properties`: (ModelProperties, optional): Defaults to None. The `ModelProperties` object required to save and deploy a model.

The `properties` is an instance of the `ModelProperties` class and has the following predefined fields:

- `bucket_uri`: str
- `compartment_id`: str
- `deployment_access_log_id`: str
- `deployment_bandwidth_mbps`: int
- `deployment_instance_count`: int
- `deployment_instance_shape`: str
- `deployment_log_group_id`: str
- `deployment_predict_log_id`: str
- `deployment_memory_in_gbs`: Union[float, int]
- `deployment_ocpus`: Union[float, int]
- `inference_conda_env`: str
- `inference_python_version`: str
- `overwrite_existing_artifact`: bool
- `project_id`: str
- `remove_existing_artifact`: bool
- `training_conda_env`: str

- `training_id`: str
- `training_python_version`: str
- `training_resource_id`: str
- `training_script_path`: str

By default, `properties` is populated from the environment variables when not specified. For example, in notebook sessions the environment variables are preset and stored in project id (`PROJECT_OCID`) and compartment id (`NB_SESSION_COMPARTMENT_OCID`). So `properties` populates these environment variables, and uses the values in methods such as `.save()` and `.deploy()`. Pass in values to overwrite the defaults. When you use a method that includes an instance of `properties`, then `properties` records the values that you pass in. For example, when you pass `inference_conda_env` into the `.prepare()` method, then `properties` records the value. To reuse the `properties` file in different places, you can export the `properties` file using the `.to_yaml()` method then reload it into a different machine using the `.from_yaml()` method.

### 12.5.7.3 Summary Status

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's Step column displays a Status of Done for the initiate step. And the Details column explains what the initiate step did such as generating a `score.py` file. The Step column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The Status column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

### 12.5.7.4 Example

```
import logging
import tempfile

from ads.automl.driver import AutoML
from ads.automl.provider import OracleAutoMLProvider
from ads.common.model_metadata import UseCaseType
from ads.dataset.dataset_browser import DatasetBrowser
from ads.model.framework.automl_model import AutoMLModel

ds = DatasetBrowser.sklearn().open("wine").set_target("target")
train, test = ds.train_test_split(test_size=0.1, random_state = 42)

ml_engine = OracleAutoMLProvider(n_jobs=-1, loglevel=logging.ERROR)
oracle_automl = AutoML(train, provider=ml_engine)
model, baseline = oracle_automl.train(
    model_list=['LogisticRegression', 'DecisionTreeClassifier'],
    random_state = 42,
    time_budget = 500
)

artifact_dir = tempfile.mkdtemp()
```

(continues on next page)

(continued from previous page)

```
automl_model = AutoMLModel(estimator=model, artifact_dir=artifact_dir)
automl_model.prepare(
    inference_conda_env="generalml_p38_cpu_v1",
    training_conda_env="generalml_p38_cpu_v1",
    use_case_type=UseCaseType.BINARY_CLASSIFICATION,
    X_sample=test.X,
    force_overwrite=True,
    training_id=None
)
automl_model.verify(test.X.iloc[:10])
model_id = automl_model.save(display_name='Demo AutoMLModel model')
deploy = automl_model.deploy(display_name='Demo AutoMLModel deployment')
automl_model.predict(test.X.iloc[:10])
automl_model.delete_deployment(wait_for_completion=True)
```

## 12.5.8 Other Frameworks

See [API Documentation](#)

### 12.5.8.1 Overview

The `ads.model.generic_model.GenericModel` class in ADS provides an efficient way to serialize almost any model class. This section demonstrates how to use the `GenericModel` class to prepare model artifacts, verify models, save models to the model catalog, deploy models, and perform predictions on model deployment endpoints.

The `GenericModel` class works with any unsupported model framework that has a `.predict()` method. For the most common model classes such as scikit-learn, XGBoost, LightGBM, TensorFlow, and PyTorch, and AutoML, we recommend that you use the ADS provided, framework-specific serializations models. For example, for a scikit-learn model, use `SKLearnmodel`. For other models, use the `GenericModel` class.

The `.verify()` method simulates a model deployment by calling the `load_model()` and `predict()` methods in the `score.py` file. With the `.verify()` method, you can debug your `score.py` file without deploying any models. The `.save()` method deploys a model artifact to the model catalog. The `.deploy()` method deploys a model to a REST endpoint.

These simple steps take your trained model and will deploy it into production with just a few lines of code.

### 12.5.8.2 Prepare Model Artifact

Instantiate a `GenericModel()` object by giving it any model object. It accepts the following parameters:

- `artifact_dir`: `str`: Artifact directory to store the files needed for deployment.
- `auth`: (`Dict`, optional): Defaults to `None`. The default authentication is set using the `ads.set_auth` API. To override the default, use `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()` and create the appropriate authentication signer and the `**kwargs` required to instantiate the `IdentityClient` object.
- `estimator`: (`Callable`): Trained model.
- `properties`: (`ModelProperties`, optional): Defaults to `None`. `ModelProperties` object required to save and deploy the model.

- `serialize: (bool, optional)`: Defaults to True. If True the model will be serialized into a pickle file. If False, you must set the `model_file_name` in the `.prepare()` method, serialize the model manually, and save it in the `artifact_dir`. You will also need to update the `score.py` file to work with this model.

The `properties` is an instance of the `ModelProperties` class and has the following predefined fields:

- `bucket_uri`: str
- `compartment_id`: str
- `deployment_access_log_id`: str
- `deployment_bandwidth_mbps`: int
- `deployment_instance_count`: int
- `deployment_instance_shape`: str
- `deployment_log_group_id`: str
- `deployment_predict_log_id`: str
- `deployment_memory_in_gbs`: Union[float, int]
- `deployment_ocpus`: Union[float, int]
- `inference_conda_env`: str
- `inference_python_version`: str
- `overwrite_existing_artifact`: bool
- `project_id`: str
- `remove_existing_artifact`: bool
- `training_conda_env`: str
- `training_id`: str
- `training_python_version`: str
- `training_resource_id`: str
- `training_script_path`: str

By default, `properties` is populated from the environment variables when not specified. For example, in notebook sessions the environment variables are preset and stored in project id (`PROJECT_OCID`) and compartment id (`NB_SESSION_COMPARTMENT_OCID`). So `properties` populates these environment variables, and uses the values in methods such as `.save()` and `.deploy()`. Pass in values to overwrite the defaults. When you use a method that includes an instance of `properties`, then `properties` records the values that you pass in. For example, when you pass `inference_conda_env` into the `.prepare()` method, then `properties` records the value. To reuse the `properties` file in different places, you can export the `properties` file using the `.to_yaml()` method then reload it into a different machine using the `.from_yaml()` method.

### 12.5.8.3 Summary Status

You can call the `.summary_status()` method after a model serialization instance such as `AutoMLModel`, `GenericModel`, `SklearnModel`, `TensorFlowModel`, or `PyTorchModel` is created. The `.summary_status()` method returns a Pandas dataframe that guides you through the entire workflow. It shows which methods are available to call and which ones aren't. Plus it outlines what each method does. If extra actions are required, it also shows those actions.

The following image displays an example summary status table created after a user initiates a model instance. The table's Step column displays a Status of Done for the initiate step. And the Details column explains what the initiate step did such as generating a `score.py` file. The Step column also displays the `prepare()`, `verify()`, `save()`, `deploy()`, and `predict()` methods for the model. The Status column displays which method is available next. After the initiate step, the `prepare()` method is available. The next step is to call the `prepare()` method.

		Actions Needed
Step	Status	Details
initiate	Done	Initiated the model
prepare()	Available	Generated runtime.yaml
		Generated score.py
		Serialized model
		Populated metadata(Custom, Taxonomy and Provenance)
verify()	Not Available	Local tested .predict from score.py
save()	Not Available	Conducted Introspect Test
		Uploaded artifact to model catalog
deploy()	Not Available	Deployed the model
predict()	Not Available	Called deployment predict endpoint

### 12.5.8.4 Example

By default, the `GenericModel` serializes to a pickle file. The following example, the user creates a model. In the prepare step, the user saves the model as a pickle file with the name `toy_model.pkl`. Then the user verifies the model, saves it to the model catalog, deploys the model and makes a prediction. Finally, the user deletes the model deployment and then deletes the model.

```
import tempfile
from ads.model.generic_model import GenericModel

class Toy:
    def predict(self, x):
        return x ** 2
model = Toy()

generic_model = GenericModel(estimator=model, artifact_dir=tempfile.mkdtemp())
generic_model.summary_status()
```

(continues on next page)



(continued from previous page)

```

generic_model.prepare(
    inference_conda_env="dbexp_p38_cpu_v1",
    model_file_name="toy_model.pkl",
    force_overwrite=True
)

# Check if the artifacts are generated correctly.
# The verify method invokes the ``predict`` function defined inside ``score.py`` in the
↳ artifact_dir
generic_model.verify(2)

# Register the model
model_id = generic_model.save(display_name="Custom Model")

# Deploy and create an endpoint for the XGBoost model
generic_model.deploy(
    display_name="My Custom Model",
    deployment_log_group_id="ocidl.loggroup.oc1.xxx.xxxxx",
    deployment_access_log_id="ocidl.log.oc1.xxx.xxxxx",
    deployment_predict_log_id="ocidl.log.oc1.xxx.xxxxx",
)

print(f"Endpoint: {generic_model.model_deployment.url}")

# Generate prediction by invoking the deployed endpoint
generic_model.predict(2)

# To delete the deployed endpoint uncomment the line below
# generic_model.delete_deployment(wait_for_completion=True)

```

You can also use the shortcut `.prepare_save_deploy()` instead of calling `.prepare()`, `.save()` and `.deploy()` separately.

```

import tempfile
from ads.model.generic_model import GenericModel

class Toy:
    def predict(self, x):
        return x ** 2
estimator = Toy()

model = GenericModel(estimator=estimator)
model.summary_status()

# If you are running the code inside a notebook session and using a service pack,
↳ `inference_conda_env` can be omitted.
model.prepare_save_deploy(inference_conda_env="dbexp_p38_cpu_v1")
model.verify(2)

# Generate prediction by invoking the deployed endpoint
model.predict(2)

```

(continues on next page)

(continued from previous page)

```
# To delete the deployed endpoint uncomment the line below  
# model.delete_deployment(wait_for_completion=True)
```

## APACHE SPARK

---

### DataFlow

Oracle Cloud Infrastructure (OCI) Data Flow is a fully managed, serverless, and on-demand Apache Spark Service that performs data processing or model training tasks on extremely large datasets without infrastructure to deploy or manage.

#### Getting Started with Data Flow

---

ads integrates with OCI Data Flow allowing users to submit and monitor Spark jobs from Notebook Sessions or from their local machine.

## 13.1 Quick Start

Data Flow is a hosted Apache Spark server. It is quick to start, and can scale to handle large datasets in parallel. ADS provides a convenient API for creating and maintaining workloads on Data Flow.

### 13.1.1 Submit a Dummy Python Script to DataFlow

#### 13.1.1.1 From a Python Environment

Submit a python script to DataFlow entirely from your python environment. The following snippet uses a dummy python script that prints “Hello World” followed by the spark version, 3.2.1.

```
from ads.jobs import DataFlow, Job, DataFlowRuntime
from uuid import uuid4
import os
import tempfile

with tempfile.TemporaryDirectory() as td:
    with open(os.path.join(td, "script.py"), "w") as f:
        f.write("""
import pyspark

def main():
    print("Hello World")
    print("Spark version is", pyspark.__version__)
```

(continues on next page)

(continued from previous page)

```

if __name__ == "__main__":
    main()
    """
)
name = f"dataflow-app-{str(uuid4())}"
dataflow_configs = (
    DataFlow()
    .with_compartment_id("oci.xx.<compartment_id>")
    .with_logs_bucket_uri("oci://<mybucket>@<mynamespace>/<dataflow-logs-
↪prefix>")
    .with_driver_shape("VM.Standard.E4.Flex")
    .with_driver_shape_config(ocpus=2, memory_in_gbs=32)
    .with_executor_shape("VM.Standard.E4.Flex")
    .with_executor_shape_config(ocpus=4, memory_in_gbs=64)
    .with_spark_version("3.2.1")
)
runtime_config = (
    DataFlowRuntime()
    .with_script_uri(os.path.join(td, "script.py"))
    .with_script_bucket("oci://<mybucket>@<mynamespace>/<subdir_to_put_and_
↪get_script>")
)
df = Job(name=name, infrastructure=dataflow_configs, runtime=runtime_config)
df.create()
df_run = df.run()

```

### 13.1.1.2 From the Command Line

The same result can be achieved from the command line using ads CLI and a yaml file.

Assuming you have the following two files written in your current directory as `script.py` and `dataflow.yaml` respectively:

```

# script.py
import pyspark
def main():
    print("Hello World")
    print("Spark version is", pyspark.__version__)
if __name__ == "__main__":
    main()

```

```

# dataflow.yaml
kind: job
spec:
  name: dataflow-app-<uuid>
  infrastructure:
    kind: infrastructure
    spec:
      compartmentId: oci.xx.<compartment_id>
      logsBucketUri: oci://<mybucket>@<mynamespace>/<dataflow-logs-

```

(continues on next page)

(continued from previous page)

```

↪prefix>
                                driverShape: VM.Standard.E4.Flex
driverShapeConfig:
  ocpus: 2
  memory_in_gbs: 32
executorShape: VM.Standard.E4.Flex
executorShapeConfig:
  ocpus: 4
  memory_in_gbs: 64
                                sparkVersion: 3.2.1
                                numExecutors: 1
                                type: dataFlow
runtime:
  kind: runtime
  spec:
                                scriptUri: script.py
                                scriptBucket: oci://<mybucket>@<mynamespace>/<subdir_to_put_and_
↪get_script>

```

```
ads jobs run -f dataflow.yaml
```

### 13.1.2 Real Data Flow Example with Conda Environment

From PySpark v3.0.0 and onwards, Data Flow allows a published conda environment as the [Spark runtime environment](#) when built with ADS. Data Flow supports published conda environments only. Conda packs are tar'd conda environments. When you publish your own conda packs to object storage, ensure that the DataFlow Resource has access to read the object or bucket. Below is a more built-out example using conda packs:

#### 13.1.2.1 From a Python Environment

```

from ads.jobs import DataFlow, Job, DataFlowRuntime
from uuid import uuid4
import os
import tempfile

with tempfile.TemporaryDirectory() as td:
    with open(os.path.join(td, "script.py"), "w") as f:
        f.write(
"""
from pyspark.sql import SparkSession
import click

@click.command()
@click.argument("app_name")
@click.option(
    "--limit", "-l", help="max number of row to print", default=10, required=False
)
@click.option("--verbose", "-v", help="print out result in verbose mode", is_flag=True)
def main(app_name, limit, verbose):

```

(continues on next page)

(continued from previous page)

```

Create a Spark session
spark = SparkSession.builder.appName(app_name).getOrCreate()

Load a csv file from dataflow public storage
df = (
    spark.read.format("csv")
        .option("header", "true")
        .option("multiLine", "true")
        .load(
            "oci://oow_2019_dataflow_lab@bigdatadatasciencelarge/usercontent/
↪kaggle_berlin_airbnb_listings_summary.csv"
        )
    )

Create a temp view and do some SQL operations
df.createOrReplaceTempView("berlin")
query_result_df = spark.sql(
    """
    SELECT
        city,
        zipcode,
        CONCAT(latitude,',', longitude) AS lat_long
    FROM berlin
    """
).limit(limit)

# Convert the filtered Spark DataFrame into JSON format
# Note: we are writing to the spark stdout log so that we can retrieve the log
↪later at the end of the notebook.
if verbose:
    rows = query_result_df.toJSON().collect()
    for i, row in enumerate(rows):
        print(f"record {i}")
        print(row)

if __name__ == "__main__":
    main()
"""
)
name = f"dataflow-app-{str(uuid4())}"
dataflow_configs = (
    DataFlow()
        .with_compartment_id("oci.xx.<compartment_id>")
        .with_logs_bucket_uri("oci://<mybucket>@<mynamespace>/<dataflow-logs-
↪prefix>")
        .with_driver_shape("VM.Standard.E4.Flex")
        .with_driver_shape_config(ocpus=2, memory_in_gbs=32)
        .with_executor_shape("VM.Standard.E4.Flex")
        .with_executor_shape_config(ocpus=4, memory_in_gbs=64)
        .with_spark_version("3.2.1")
    )
runtime_config = (

```

(continues on next page)

(continued from previous page)

```

        DataFlowRuntime()
        .with_script_uri(os.path.join(td, "script.py"))
        .with_script_bucket("oci://<mybucket>@<mynamespace>/<subdir_to_put_and_
↪get_script>")
        .with_custom_conda(uri="oci://<mybucket>@<mynamespace>/<path_to_conda_
↪pack>")
        .with_arguments(["run-test", "-v", "-l", "5"])
    )
    df = Job(name=name, infrastructure=dataflow_configs, runtime=runtime_config)
    df.create()
    df_run = df.run()

```

### 13.1.2.2 From the Command Line

Again, assume you have the following two files written in your current directory as `script.py` and `dataflow.yaml` respectively:

```

# script.py
from pyspark.sql import SparkSession
import click

@click.command()
@click.argument("app_name")
@click.option(
    "--limit", "-l", help="max number of row to print", default=10, required=False
)
@click.option("--verbose", "-v", help="print out result in verbose mode", is_flag=True)
def main(app_name, limit, verbose):
    Create a Spark session
    spark = SparkSession.builder.appName(app_name).getOrCreate()

    Load a csv file from dataflow public storage
    df = (
        spark.read.format("csv")
        .option("header", "true")
        .option("multiLine", "true")
        .load(
            "oci://oow_2019_dataflow_lab@bigdatadatasciencelarge/usercontent/
↪kaggle_berlin_airbnb_listings_summary.csv"
        )
    )

    Create a temp view and do some SQL operations
    df.createOrReplaceTempView("berlin")
    query_result_df = spark.sql(
        """
        SELECT
            city,
            zipcode,
            CONCAT(latitude,',', longitude) AS lat_long
        FROM berlin

```

(continues on next page)

(continued from previous page)

```

"""
).limit(limit)

# Convert the filtered Spark DataFrame into JSON format
# Note: we are writing to the spark stdout log so that we can retrieve the log
↳ later at the end of the notebook.
if verbose:
    rows = query_result_df.toJSON().collect()
    for i, row in enumerate(rows):
        print(f"record {i}")
        print(row)

if __name__ == "__main__":
    main()

```

```

# dataflow.yaml
kind: job
spec:
  name: dataflow-app-<uuid>
  infrastructure:
    kind: infrastructure
    spec:
      compartmentId: oci.xx.<compartment_id>
      logsBucketUri: oci://<mybucket>@<mynamespace>/<dataflow-logs-
↳ prefix>

      driverShape: VM.Standard.E4.Flex
      driverShapeConfig:
        ocpu: 2
        memory_in_gbs: 32
      executorShape: VM.Standard.E4.Flex
      executorShapeConfig:
        ocpu: 4
        memory_in_gbs: 64
      sparkVersion: 3.2.1
      numExecutors: 1
    type: dataFlow
  runtime:
    kind: runtime
    spec:
      scriptUri: script.py
      scriptBucket: oci://<mybucket>@<mynamespace>/<subdir_to_put_and_
↳ get_script>

      conda:
        uri: oci://<mybucket>@<mynamespace>/<path_to_conda_pack>
        type: published
      args:
        - "run-test"
        - "-v"
        - "-l"
        - "5"

```



```
ads jobs run -f dataflow.yaml
```

## 13.2 Setup and Installation

### 13.2.1 Notebook Session Development

Follow these set up instructions to submit Spark Jobs to Data Flow from an OCI Notebook Session.

#### 13.2.1.1 Pyspark Environment

To setup PySpark environment, install one of the PySpark conda environments from the [Environment Explorer](#)  
Example -

```
odsc conda install -s pyspark30_p37_cpu_v5
```

Find the information about the latest pyspark conda environment [here](#)

Activate the conda environment to upgrade to the latest oracle-ads

```
conda activate /home/datascience/conda/pyspark30_p37_cpu_v5
pip install oracle-ads[data_science, data, opctl] --upgrade
```

#### 13.2.1.2 Configuring core-site.xml

When the conda environment is installed, a templated version of *core-site.xml* is also installed. You can update the *core-site.xml* file using an automated configuration or manually.

##### Authentication with Resource Principals

Authentication to Object Storage can be done with a resource principal.

For automated configuration, run the following command in a terminal

```
odsc core-site config -a resource_principal
```

This command will populate the file `~/spark_conf_dir/core-site.xml` with the values needed to connect to Object Storage.

The following command line options are available:

- *-a, --authentication* Authentication mode. Supports *resource\_principal* and *api\_key* (default).
- *-r, --region* Name of the region.
- *-o, --overwrite* Overwrite *core-site.xml*.
- *-O, --output* Output path for *core-site.xml*.
- *-q, --quiet* Suppress non-error output.
- *-h, --help* Show help message and exit.

To manually configure the `core-site.xml` file, you edit the file, and then specify these values:

`fs.oci.client.hostname:` The Object Storage endpoint for your region. See <https://docs.oracle.com/iaas/api/#/en/objectstorage/20160918/> for available endpoints.

`fs.oci.client.custom.authenticator:` Set the value to `com.oracle.bmc.hdfs.auth.ResourcePrincipalsCustomAuthenticator`.

When using resource principals, these properties don't need to be configured:

- `fs.oci.client.auth.tenantId`
- `fs.oci.client.auth.userId`
- `fs.oci.client.auth.fingerprint`
- `fs.oci.client.auth.pemfilepath`

The following example `core-site.xml` file illustrates using resource principals for authentication to access Object Storage:

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>fs.oci.client.hostname</name>
    <value>https://objectstorage.us-ashburn-1.oraclecloud.com</value>
  </property>
  <property>
    <name>fs.oci.client.custom.authenticator</name>
    <value>com.oracle.bmc.hdfs.auth.ResourcePrincipalsCustomAuthenticator</value>
  </property>
</configuration>
```

For details, see [HDFS connector for Object Storage #using resource principals for authentication](#).

### Authentication with API Keys

When using authentication with **API keys**, the `core-site.xml` file is be updated in two ways, automated or manual configuration.

For automated configuration, you use the `odsc` command line tool. With an OCI configuration file, you can run

```
odsc core-site config -o
```

By default, this command uses the OCI configuration file stored in `~/.oci/config`, automatically populates the `core-site.xml` file, and then saves it to `~/spark_conf_dir/core-site.xml`.

The following command line options are available:

- `-a, --authentication` Authentication mode. Supports `resource_principal` and `api_key` (default).
- `-c, --configuration` Path to the OCI configuration file.
- `-p, --profile` Name of the profile.
- `-r, --region` Name of the region.
- `-o, --overwrite` Overwrite `core-site.xml`.
- `-O, --output` Output path for `core-site.xml`.
- `-q, --quiet` Suppress non-error output.
- `-h, --help` Show help message and exit.

To manually configure the `core-site.xml` file, you must specify these parameters:

`fs.oci.client.hostname`: Address of Object Storage. For example, `https://objectstorage.us-ashburn-1.oraclecloud.com`. You must replace `us-ashburn-1` with the region you are in. `fs.oci.client.auth.tenantId`: OCID of your tenancy. `fs.oci.client.auth.userId`: Your user OCID. `fs.oci.client.auth.fingerprint`: Fingerprint for the key pair. `fs.oci.client.auth.pemfilepath`: The fully qualified file name of the private key used for authentication.

The values of these parameters are found in the OCI configuration file.

## 13.2.2 Local Development

Follow these set up instructions to submit Spark Jobs to Data Flow from your local machine.

### 13.2.2.1 PySpark Environment

#### Prerequisite

You have completed *Local Development Environment Setup*

Use ADS CLI to setup a PySpark conda environment. Currently, the ADS CLI only supports fetching conda packs published by you. If you haven't already published a conda pack, you can create one using ADS CLI

To install from your published environment source -

```
ads conda install oci://mybucket@mynamespace/path/to/pyspark/env
```

To create a conda pack for your local use -

```
cat <<EOF> pyspark.yaml

dependencies:
  - pyspark
  - pip
  - pip:
    - oracle-ads
name: pysparkenv
EOF
```

```
ads create -f pyspark.yaml
```

```
ads publish -s pysparkenv
```

### 13.2.2.2 Developing in Visual Studio Code

#### Prerequisites

1. Setup Visual Studio Code development environment by following steps from *Local Development Environment Setup*
2. `ads conda install <oci uri of pyspark conda environment>`. Currently, we cannot access service pack directly. You can instead publish a pyspark service pack to your object storage and use the URI for the pack in OCI Object Storage.

Once the development environment is setup, you could write your code and run it from the terminal of the Visual Studio Code.

core-site.xml is setup automatically when you install a pyspark conda pack.

### 13.2.3 Logging From DataFlow

If using the ADS Python SDK,

To create and run a Data Flow application, you must specify a compartment and a bucket for storing logs under the same compartment:

```
compartment_id = "<compartment_id>"
logs_bucket_uri = "<logs_bucket_uri>"
```

Ensure that you set up the correct policies. For instance, for Data Flow to access logs bucket, use a policy like:

```
ALLOW SERVICE dataflow TO READ objects IN tenancy WHERE target.bucket.name='dataflow-logs'
↪ '
```

For more information, see the [Data Flow documentation](#).

## 13.3 Running your Spark Application on OCI Data Flow

Submit your code to DataFlow for workloads that require larger resources.

### 13.3.1 Notebook Extension

For most Notebook users, local or OCI Notebook Sessions, the notebook extension is the most straightforward integration with dataflow. It's a "Set It and Forget It" API with options to update ad-hoc. You can configure your dataflow runs by running `ads opctl configure` in the terminal.

After setting up your dataflow config, you can return to the Notebook. Import `ads` and `DataFlowConfig`:

```
import ads
from ads.jobs.utils import DataFlowConfig
```

Load the dataflow extension inside the notebook cell -

```
%load_ext ads.jobs.extension
```

Define config. If you have not yet configured your dataflow setting, or would like to amend the defaults, you can modify as shown below:

```
dataflow_config = DataFlowConfig()
dataflow_config.compartment_id = "ocid1.compartment.<your compartment ocid>"
dataflow_config.driver_shape = "VM.Standard.E4.Flex"
dataflow_config.driver_shape_config = oci.data_flow.models.ShapeConfig(ocpus=2, memory_
↪ in_gbs=32)
dataflow_config.executor_shape = "VM.Standard.E4.Flex"
dataflow_config.executor_shape_config = oci.data_flow.models.ShapeConfig(ocpus=4, memory_
↪ in_gbs=64)
```

(continues on next page)

(continued from previous page)

```
dataflow_config.logs_bucket_uri = "oci://<my-bucket>@<my-tenancy>/"
dataflow_config.spark_version = "3.2.1"
dataflow_config.configuration = {"spark.driver.memory": "512m"}
```

Use the config defined above to submit the cell.

### Tip

Get more information about the dataflow extension by running `%dataflow -h`

Call the dataflow magic command in the first line of your cell to run it on dataflow.

```
%%dataflow run -f a_script.py -c {dataflow_config} -w -o -- abc -l 5 -v
```

This header will: - save the cell as a file called `script.py` and store it in your `dataflow_config.script_bucket`  
 - After the `--` notation, all parameters are sent to your script. For example `abc` is a positional argument, and `l` and `v` are named arguments.

Below is a full example:

```
%%dataflow run -f a_script.py -c {dataflow_config} -w -o -- abc -l 5 -v
from pyspark.sql import SparkSession
import click

@click.command()
@click.argument("app_name")
@click.option(
    "--limit", "-l", help="max number of rows to print", default=10, required=False
)
@click.option("--verbose", "-v", help="print out result in verbose mode", is_flag=True)
def main(app_name, limit, verbose):
    # Create a Spark session
    spark = SparkSession.builder.appName(app_name).getOrCreate()

    # Load a csv file from dataflow public storage
    df = (
        spark.read.format("csv")
        .option("header", "true")
        .option("multiLine", "true")
        .load(
            "oci://oow_2019_dataflow_lab@bigdatadatasciencelarge/usercontent/kaggle_
            berlin_airbnb_listings_summary.csv"
        )
    )

    # Create a temp view and do some SQL operations
    df.createOrReplaceTempView("berlin")
    query_result_df = spark.sql(
        """
        SELECT
            city,
```

(continues on next page)

(continued from previous page)

```

        zipcode,
        CONCAT(latitude,',', longitude) AS lat_long
    FROM berlin
    """
).limit(limit)

# Convert the filtered Spark DataFrame into JSON format
# Note: we are writing to the spark stdout log so that we can retrieve the log later.
↳ at the end of the notebook.
if verbose:
    rows = query_result_df.toJSON().collect()
    for i, row in enumerate(rows):
        print(f"record {i}")
        print(row)

if __name__ == "__main__":
    main()

```

### 13.3.2 ADS CLI

#### Prerequisites

1. *Install ADS CLI*
2. *Configure Defaults*

#### Tip

If, for some reason, you are unable to use CLI, instead skip to the [Create, Run Data Flow Application Using ADS Python SDK](#) section below.

Sometimes your code is too complex to run in a single cell, and it's better run as a notebook or file. In that case, use the ADS Opctl CLI.

To submit your notebook to DataFlow using the ads CLI, run:

```
ads opctl run -s <folder where notebook is located> -e <notebook name> -b dataflow
```

#### Tip

You can avoid running cells that are not DataFlow environment compatible by tagging the cells and then providing the tag names to ignore. In the following example cells that are tagged `ignore` and `remove` will be ignored - `--exclude-tag ignore --exclude-tag remove`

#### Tip

You can run the notebook in your local pyspark environment before submitting to DataFlow using the same CLI with `-b local`

```
# Activate the Pyspark conda environment in local
ads opctl run -s <notebook directory> -e <notebook file> -b local
```

You could submit a notebook using ADS SDK APIs. Here is an example to submit a notebook -

```
from ads.jobs import Job, DataFlow, DataFlowNotebookRuntime

df = (
    DataFlow()
    .with_compartment_id(
        "ocidl.compartment.oc1..
↪aaaaaaaapvb3hearqum6wjvlcpzm5ptfxqa7xfftpt4h72xx46ygavkqteq"
    )
    .with_driver_shape("VM.Standard.E4.Flex")
    .with_driver_shape_config(ocpus=2, memory_in_gbs=32)
    .with_executor_shape("VM.Standard.E4.Flex")
    .with_executor_shape_config(ocpus=4, memory_in_gbs=64)
    .with_logs_bucket_uri("oci://mybucket@mytenancy/")
)
rt = (
    DataFlowNotebookRuntime()
    .with_notebook(
        "<path to notebook>"
    ) # This could be local path or http path to notebook ipynb file
    .with_script_bucket("<my-bucket>")
    .with_exclude_tag(["ignore", "remove"]) # Cells to Ignore
)
job = Job(infrastructure=df, runtime=rt).create(overwrite=True)
df_run = job.run(wait=True)
```

### 13.3.3 ADS Python SDK

To create a Data Flow application using the ADS Python API you need two components:

- DataFlow, a subclass of Infrastructure.
- DataFlowRuntime, a subclass of Runtime.

DataFlow stores properties specific to Data Flow service, such as compartment\_id, logs\_bucket\_uri, and so on. You can set them using the with\_{property} functions:

- with\_compartment\_id
- with\_configuration
- with\_driver\_shape
- with\_driver\_shape\_config
- with\_executor\_shape
- with\_executor\_shape\_config
- with\_language
- with\_logs\_bucket\_uri

- `with_metastore_id` ([doc](#))
- `with_num_executors`
- `with_spark_version`
- `with_warehouse_bucket_uri`

For more details, see [DataFlow class documentation](#).

DataFlowRuntime stores properties related to the script to be run, such as the path to the script and CLI arguments. Likewise all properties can be set using `with_{property}`. The DataFlowRuntime properties are:

- `with_script_uri`
- `with_script_bucket`
- `with_archive_uri` ([doc](#))
- `with_archive_bucket`
- `with_custom_conda`

For more details, see the [runtime class documentation](#).

Since service configurations remain mostly unchanged across multiple experiments, a DataFlow object can be reused and combined with various DataFlowRuntime parameters to create applications.

In the following “hello-world” example, DataFlow is populated with `compartment_id`, `driver_shape`, `driver_shape_config`, `executor_shape`, `executor_shape_config` and `spark_version`. DataFlowRuntime is populated with `script_uri` and `script_bucket`. The `script_uri` specifies the path to the script. It can be local or remote (an Object Storage path). If the path is local, then `script_bucket` must be specified additionally because Data Flow requires a script to be available in Object Storage. ADS performs the upload step for you, as long as you give the bucket name or the Object Storage path prefix to upload the script. Either can be given to `script_bucket`. For example, either `with_script_bucket("<bucket_name>")` or `with_script_bucket("oci://<bucket_name>@<namespace>/<prefix>")` is accepted. In the next example, the prefix is given for `script_bucket`.

```
from ads.jobs import DataFlow, Job, DataFlowRuntime
from uuid import uuid4
import os
import tempfile

with tempfile.TemporaryDirectory() as td:
    with open(os.path.join(td, "script.py"), "w") as f:
        f.write(
            """
import pyspark

def main():
    print("Hello World")
    print("Spark version is", pyspark.__version__)

if __name__ == "__main__":
    main()
            """
        )
    name = f"dataflow-app-{str(uuid4())}"
    dataflow_configs = (
        DataFlow()
```

(continues on next page)



(continued from previous page)

```

.with_compartment_id("oci.xx.<compartment_id>")
.with_logs_bucket_uri("oci://mybucket@mynamespace/dflogs")
.with_driver_shape("VM.Standard.E4.Flex")
    .with_driver_shape_config(ocpus=2, memory_in_gbs=32)
    .with_executor_shape("VM.Standard.E4.Flex")
    .with_executor_shape_config(ocpus=4, memory_in_gbs=64)
.with_spark_version("3.0.2")
)
runtime_config = (
    DataFlowRuntime()
    .with_script_uri(os.path.join(td, "script.py"))
    .with_script_bucket("oci://mybucket@namespace/prefix")
    .with_custom_conda("oci://<mybucket>@<mynamespace>/<path/to/conda_pack>")
)
df = Job(name=name, infrastructure=dataflow_configs, runtime=runtime_config)
df.create()

```

To run this application, you could use:

```
df_run = df.run()
```

After the run completes, check the stdout log from the application by running:

```
print(df_run.logs.application.stdout)
```

You should this in the log:

```

Hello World
Spark version is 3.0.2

```

### Note on Policy

```
ALLOW SERVICE dataflow TO READ objects IN tenancy WHERE target.bucket.name='dataflow-logs'
```

Data Flow supports adding third-party libraries using a ZIP file, usually called `archive.zip`, see the [Data Flow documentation](#) about how to create ZIP files. Similar to scripts, you can specify an archive ZIP for a Data Flow application using `with_archive_uri`. In the next example, `archive_uri` is given as an Object Storage location. `archive_uri` can also be local so you must specify `with_archive_bucket` and follow the same rule as `with_script_bucket`.

```

from ads.jobs import DataFlow, DataFlowRun, DataFlowRuntime, Job
from uuid import uuid4
import tempfile
import os

with tempfile.TemporaryDirectory() as td:
    with open(os.path.join(td, "script.py"), "w") as f:
        f.write(
            """
from pyspark.sql import SparkSession
import click

```

(continues on next page)

(continued from previous page)

```

@click.command()
@click.argument("app_name")
@click.option(
    "--limit", "-l", help="max number of row to print", default=10, required=False
)
@click.option("--verbose", "-v", help="print out result in verbose mode", is_flag=True)
def main(app_name, limit, verbose):
    # Create a Spark session
    spark = SparkSession.builder.appName(app_name).getOrCreate()

    # Load a csv file from dataflow public storage
    df = (
        spark.read.format("csv")
        .option("header", "true")
        .option("multiLine", "true")
        .load(
            "oci://oow_2019_dataflow_lab@bigdatadatasciencelarge/usercontent/kaggle_
↳berlin_airbnb_listings_summary.csv"
        )
    )

    # Create a temp view and do some SQL operations
    df.createOrReplaceTempView("berlin")
    query_result_df = spark.sql(
        """
        SELECT
            city,
            zipcode,
            CONCAT(latitude,',', longitude) AS lat_long
        FROM berlin
        """
    ).limit(limit)

    # Convert the filtered Spark DataFrame into JSON format
    # Note: we are writing to the spark stdout log so that we can retrieve the log later.
↳at the end of the notebook.
    if verbose:
        rows = query_result_df.toJSON().collect()
        for i, row in enumerate(rows):
            print(f"record {i}")
            print(row)

if __name__ == "__main__":
    main()
    """
)

name = f"dataflow-app-{str(uuid4())}"
dataflow_configs = (
    DataFlow()
    .with_compartment_id("oci1.xxx.<compartment_ocid>")

```

(continues on next page)

(continued from previous page)

```

.with_logs_bucket_uri("oci://mybucket@mynamespace/prefix")
.with_driver_shape("VM.Standard.E4.Flex")
    .with_driver_shape_config(ocpus=2, memory_in_gbs=32)
    .with_executor_shape("VM.Standard.E4.Flex")
    .with_executor_shape_config(ocpus=4, memory_in_gbs=64)
.with_spark_version("3.0.2")
)
runtime_config = (
    DataFlowRuntime()
    .with_script_uri(os.path.join(td, "script.py"))
    .with_script_bucket("oci://<bucket>@<namespace>/prefix/path")
    .with_archive_uri("oci://<bucket>@<namespace>/prefix/archive.zip")
    .with_custom_conda(uri="oci://<mybucket>@<mynamespace>/<my-conda-uri>")
)
df = Job(name=name, infrastructure=dataflow_configs, runtime=runtime_config)
df.create()

```

You can pass arguments to a Data Flow run as a list of strings:

```
df_run = df.run(args=["run-test", "-v", "-l", "5"])
```

You can save the application specification into a YAML file for future reuse. You could also use the json format.

```
print(df.to_yaml("sample-df.yaml"))
```

You can also load a Data Flow application directly from the YAML file saved in the previous example:

```
df2 = Job.from_yaml(uri="sample-df.yaml")
```

Creating a new job and a run:

```
df_run2 = df2.create().run()
```

Deleting a job cancels associated runs:

```
df2.delete()
df_run2.status
```

You can also load a Data Flow application from an OCID:

```
df3 = Job.from_dataflow_job(df.id)
```

Creating a run under the same application:

```
df_run3 = df3.run()
```

Now there are 2 runs under the df application:

```
assert len(df.run_list()) == 2
```

When you run a Data Flow application, a DataFlowRun object is created. You can check the status, wait for a run to finish, check its logs afterwards, or cancel a run in progress. For example:

```
df_run.status
df_run.wait()
```

watch is an alias of wait, so you can also call `df_run.watch()`.

There are three types of logs for a run:

- application log
- driver log
- executor log

Each log consists of `stdout` and `stderr`. For example, to access `stdout` from application log, you could use:

```
df_run.logs.application.stdout
```

Then you could check it with:

```
df_run.logs.application.stderr
df_run.logs.executor.stdout
df_run.logs.executor.stderr
```

You can also examine head or tail of the log, or download it to a local path. For example,

```
log = df_run.logs.application.stdout
log.head(n=1)
log.tail(n=1)
log.download(<local-path>)
```

For the sample script, the log prints first five rows of a sample dataframe in JSON and it looks like:

```
record 0
{"city":"Berlin","zipcode":"10119","lat_long":"52.53453732241747,13.402556926822387"}
record 1
{"city":"Berlin","zipcode":"10437","lat_long":"52.54851279221664,13.404552826587466"}
record 2
{"city":"Berlin","zipcode":"10405","lat_long":"52.534996191586714,13.417578665333295"}
record 3
{"city":"Berlin","zipcode":"10777","lat_long":"52.498854933130026,13.34906453348717"}
record 4
{"city":"Berlin","zipcode":"10437","lat_long":"52.5431572633131,13.415091104515707"}
```

Calling `log.head(n=1)` returns this:

```
'record 0'
```

Calling `log.tail(n=1)` returns this:

```
{"city":"Berlin","zipcode":"10437","lat_long":"52.5431572633131,13.415091104515707"}
```

A link to run the page in the OCI Console is given using the `run_details_link` property:

```
df_run.run_details_link
```

To list Data Flow applications, a compartment id must be given with any optional filtering criteria. For example, you can filter by name of the application:

```
Job.dataflow_job(compartment_id=compartment_id, display_name=name)
```

### 13.3.3.1 YAML

You can create a Data Flow job directly from a YAML string. You can pass a YAML string into the `Job.from_yaml()` function to build a Data Flow job:

```
kind: job
spec:
  id: <dataflow_app_ocid>
  infrastructure:
    kind: infrastructure
    spec:
      compartmentId: <compartment_id>
      driverShape: VM.Standard.E4.Flex
      driverShapeConfig:
        ocpus: 2
        memory_in_gbs: 32
      executorShape: VM.Standard.E4.Flex
      executorShapeConfig:
        ocpus: 4
        memory_in_gbs: 64
      id: <dataflow_app_ocid>
      language: PYTHON
      logsBucketUri: <logs_bucket_uri>
      numExecutors: 1
      sparkVersion: 2.4.4
    type: dataFlow
  name: dataflow_app_name
  runtime:
    kind: runtime
    spec:
      scriptBucket: bucket_name
      scriptPathURI: oci://<bucket_name>@<namespace>/<prefix>
    type: dataFlow
```

#### Data Flow Infrastructure YAML Schema

```
kind:
  allowed:
    - infrastructure
  required: true
  type: string
spec:
  required: true
  type: dict
  schema:
    compartmentId:
      required: false
      type: string
    displayName:
      required: false
      type: string
    driverShape:
      required: false
      type: string
```

(continues on next page)

(continued from previous page)

```

driverShapeConfig:
  required: false
  type: dict
  schema:
    ocpus:
      required: true
      type: float
    memory_in_gbs:
      required: true
      type: float
executorShape:
  required: false
  type: string
executorShapeConfig:
  required: false
  type: dict
  schema:
    ocpus:
      required: true
      type: float
    memory_in_gbs:
      required: true
      type: float
id:
  required: false
  type: string
language:
  required: false
  type: string
logsBucketUri:
  required: false
  type: string
metastoreId:
  required: false
  type: string
numExecutors:
  required: false
  type: integer
sparkVersion:
  required: false
  type: string
type:
  allowed:
    - dataFlow
  required: true
  type: string

```

**Data Flow Runtime YAML Schema**

```

kind:
  allowed:
    - runtime

```

(continues on next page)

(continued from previous page)

```

    required: true
    type: string
spec:
    required: true
    type: dict
    schema:
        archiveBucket:
            required: false
            type: string
        archiveUri:
            required: false
            type: string
        args:
            nullable: true
            required: false
            schema:
                type: string
            type: list
        conda:
            nullable: false
            required: false
            type: dict
            schema:
                slug:
                    required: true
                    type: string
                type:
                    allowed:
                        - service
                    required: true
                    type: string
        env:
            type: list
            required: false
            schema:
                type: dict
        freeform_tag:
            required: false
            type: dict
        scriptBucket:
            required: false
            type: string
        scriptPathURI:
            required: false
            type: string
type:
    allowed:
        - dataFlow
    required: true
    type: string

```

## 13.4 spark-defaults.conf

The `spark-defaults.conf` file is used to define the properties that are used by Spark. This file can be configured manually or with the aid of the `odsc` command-line tool. The best practice is to use the `odsc data-catalog config` command-line tool when you want to connect to Data Catalog. It gathers information about your environment and uses that to build the file.

The `odsc data-catalog config` command-line tool uses the `--metastore` option to define the Data Catalog Metastore OCID. There are no required command-line options. Default values are used or values are taken from your notebook session environment and OCI configuration file. Below is a discussion of common parameters that you may need to override.

The `--authentication` option sets the authentication mode. It supports resource principal and API keys. The preferred method for authentication is resource principal and this is sent with `--authentication resource_principal`. If you want to use API keys then use the option `--authentication api_key`. If the `--authentication` is not specified, API keys will be used. When API keys are used, information from the OCI configuration file is used to create the `spark-defaults.conf` file.

The Object Storage and Data Catalog are regional services. By default, the region is set to the region that your notebook session is in. This information is taken from the environment variable `NB_REGION`. Use the `--region` option to override this behavior.

The default location of the `spark-defaults.conf` file is in the `~/spark_conf_dir` directory, as defined in the `SPARK_CONF_DIR` environment variable. Use the `--output` option to define the directory where the file is to be written.

### 13.4.1 odsc Command-line

The `odsc data-catalog config` command-line tool is ideal for setting up the `spark-defaults.conf` file as it gathers information about your environment and uses that to build the file.

You will need to determine what settings are appropriate for your configuration. However, the following will work for most configurations.

```
odsc data-catalog config --authentication resource_principal
```

If the option `--authentication api_key` is used, it will extract information from the OCI configuration file that is stored in `~/.oci/config`. Use the `--config` option to change the path and the `--profile` option to specify what OCI configuration profile will be used. The default profile is `DEFAULT`.

A default Data Catalog Metastore OCID can be set using the `--metastore` option. This value can be overridden at run-time.

```
odsc data-catalog config --authentication resource_principal --metastore <metastore_id>
```

The `<metastore_id>` must be replaced with the OCID for the Data Catalog Metastore that is to be used.

For details on the command-line option use the command:

```
odsc data-catalog config --help
```



### 13.4.2 Manual

The `odsc` command-line tool is the preferred method for configuring the `spark-defaults.conf` file. However, if you are not in a notebook session or if you have special requirements, you may need to manually configure the file. This section will guide you through the steps.

When a Data Science Conda environment is installed, it includes a template of the `spark-defaults.conf` file. The following sections provide guidance to make the required changes.

These parameters define the Object Storage address that backs the Data Catalog entry. This is the location of the data warehouse. You also need to define the address of the Data Catalog Metastore.

- `spark.hadoop.fs.oci.client.hostname`: Address of Object Storage for the data warehouse. For example, `https://objectstorage.us-ashburn-1.oraclecloud.com`. Replace `us-ashburn-1` with the region you are in.
- `spark.hadoop.oci.metastore.uris`: The address of Data Catalog Metastore. For example, `https://datacatalog.us-ashburn-1.oci.oraclecloud.com/`. Replace `us-ashburn-1` with the region you are in.

You can set a default metastore with the following parameter. This can be overridden at run time. Setting it is optional.

- `spark.hadoop.oracle.dcat.metastore.id`: The OCID of Data Catalog Metastore. For example, `ocid1.datacatalogmetastore.<unique_id>`

Depending on the authentication method that is to be used there are additional parameters that need to be set. See the following sections for guidance.

#### 13.4.2.1 Resource Principal

Update the `spark-defaults.conf` file parameters to use resource principal to authenticate:

- `spark.hadoop.fs.oci.client.custom.authenticator`: Set the value to `com.oracle.bmc.hdfs.auth.ResourcePrincipalsCustomAuthenticator`.
- `spark.hadoop.oracle.dcat.metastore.client.custom.authentication_provider`: Set the value to `com.oracle.bmc.hdfs.auth.ResourcePrincipalsCustomAuthenticator`.

#### 13.4.2.2 API Keys

Update the `spark-defaults.conf` file parameters to use API keys to authenticate:

- `spark.hadoop.OCI_FINGERPRINT_METADATA`: Fingerprint for the key pair being used.
- `spark.hadoop.OCI_PASSPHRASE_METADATA`: Passphrase used for the key if it is encrypted.
- `spark.hadoop.OCI_PVT_KEY_FILE_PATH`: The full path and file name of the private key used for authentication.
- `spark.hadoop.OCI_REGION_METADATA`: An Oracle Cloud Infrastructure region. Example: `us-ashburn-1`
- `spark.hadoop.OCI_USER_METADATA`: Your user OCID.
- `spark.hadoop.fs.oci.client.auth.fingerprint`: Fingerprint for the key pair being used.
- `spark.hadoop.fs.oci.client.auth.passphrase`: Passphrase used for the key if it is encrypted.
- `spark.hadoop.fs.oci.client.auth.pemfilepath`: The full path and file name of the private key used for authentication.
- `spark.hadoop.fs.oci.client.auth.tenantId`: OCID of your tenancy.
- `spark.hadoop.fs.oci.client.auth.userId`: Your user OCID.

- `spark.hadoop.fs.oci.client.custom.authenticator`: Set the value to `com.oracle.pic.dcat.metastore.common.auth.provider.UserPrincipalsCustomAuthenticationDetailsProvider`
- `spark.hadoop.spark.hadoop.OCI_TENANT_METADATA`: OCID of your tenancy.

The values of these parameters are found in the OCI configuration file.

## 13.5 Data Catalog Metastore

This section demonstrates how to configure the `spark-defaults.conf` file so that you can connect with the Oracle Cloud Infrastructure (OCI) [Data Catalog Metastore](#). This connection is used to run a [PySpark](#) application using [OCI Data Flow](#) and [Data Science Jobs](#). The data will be stored in [OCI Object Storage](#). Thus, you will work with data that is stored in Object Storage, information about the location and structure of that data will be managed by Data Catalog Metastore, compute will be provided by Data Flow and all of this will be run in a Job.

OCI [Data Catalog](#) is a metadata management service that helps data professionals discover data and support data governance. The [Data Catalog Metastore](#) provides schema definitions for objects in structured and unstructured data assets that reside in Object Storage. Use the metastore as a central metadata repository to manage data tables that are backed by files in Object Storage.

OCI [Data Flow](#) is a fully managed [Apache Spark](#) service. This section demonstrates how to use PySpark to create Spark applications.

[Data Science Jobs](#) allows you to run customized tasks outside of a notebook session. A Job is a template that describes a task that you want to perform. In this section, that task is to run a PySpark application using Data Flow. Since the Job is run outside of a notebook, command-line arguments can be passed to the Job such that it performs customized activities. [OCI Logging](#) is used to capture events. You can also read and write data to Object Storage directly or with the aid of Data Catalog.

Data Flow can access the Data Catalog Metastore to securely store and retrieve schema definitions for unstructured and structured data assets in Object Storage. For integration with Data Flow, the metastore provides an invocation endpoint. This endpoint is a Hive Metastore interface.

[Apache Hive](#) is a data warehousing framework that facilitates read, write, or manage operations on large datasets residing in distributed file systems. The Data Catalog Metastore is backed by the Apache Hive Metastore. A Hive Metastore is the central repository of metadata for a Hive cluster. It stores metadata for data structures such as databases, tables, and partitions in a relational database, backed by files maintained in Object Storage. [Apache Spark SQL](#) makes use of a Hive Metastore for this purpose.

### 13.5.1 Prerequisite

To access the data in the Data Catalog or work with Data Flow, there are a number of steps that need to be completed.

To configure Data Flow you will need to:

- DataFlow requires a bucket to store the logs, and a data warehouse bucket. Refer to the Data Flow documentation for [setting up storage](#).
- DataFlow requires policies to be set in IAM to access resources to manage and run applications/sessions. Refer to the Data Flow documentation on how to [setup policies](#).
- DataFlow natively supports conda packs published to OCI Object Storage. Ensure the Data Flow Resource has read access to the bucket or path of your published conda pack, and that the spark version  $\geq 3$  when running your Data Flow Application/Session.
- The `core-site.xml` file needs to be configured.

To configure Data Catalog you will need to:

- Data Catalog requires policies to be set in IAM. Refer to the Data Catalog documentation on how to [setup policies](#).
- The `spark-defaults.conf` file needs to be configured.

## 13.5.2 Quick Start

### 13.5.2.1 Data Flow

```
from ads.jobs import DataFlow, DataFlowRun, DataFlowRuntime

# Update these values
job_name = "<job_name>"
logs_bucket = "oci://<bucket_name>@<namespace>/<prefix>"
metastore_id = "<metastore_id>"
script_bucket = "oci://<bucket_name>@<namespace>/<prefix>"

compartment_id = os.environ.get("NB_SESSION_COMPARTMENT_OCID")
driver_shape = "VM.Standard.E4.Flex"
driver_shape_config = {"ocpus":2, "memory_in_gbs":32}
executor_shape = "VM.Standard.E4.Flex"
executor_shape_config = {"ocpus":4, "memory_in_gbs":64}
spark_version = "3.2.1"

# A python script to be run in Data Flow
script = '''
from pyspark.sql import SparkSession

def main():

    database_name = "employee_attrition"
    table_name = "orcl_attrition"

    # Create a Spark session
    spark = SparkSession \\\
        .builder \\\
        .appName("Python Spark SQL basic example") \\\
        .enableHiveSupport() \\\
        .getOrCreate()

    # Load a CSV file from a public Object Storage bucket
    df = spark \\\
        .read \\\
        .format("csv") \\\
        .option("header", "true") \\\
        .option("multiLine", "true") \\\
        .load("oci://hosted-ds-datasets@bigdatadatasciencelarge/synthetic/orcl_attrition.
↪ csv")

    print(f"Creating {database_name}")
    spark.sql(f"DROP DATABASE IF EXISTS {database_name} CASCADE")
    spark.sql(f"CREATE DATABASE IF NOT EXISTS {database_name}")
```

(continues on next page)

(continued from previous page)

```

# Write the data to the database
df.write.mode("overwrite").saveAsTable(f"{database_name}.{table_name}")

# Use Spark SQL to read from the database.
query_result_df = spark.sql(f"""
    SELECT EducationField, SalaryLevel, JobRole FROM
↪{database_name}.{table_name} limit 10
    """)

# Convert the filtered Apache Spark DataFrame into JSON format and write it out to
↪stdout
# so that it can be captured in the log.
print('\n'.join(query_result_df.toJSON().collect()))

if __name__ == '__main__':
    main()
'''

# Saves the python script to local path.
dataflow_base_folder = tempfile.mkdtemp()
script_uri = os.path.join(dataflow_base_folder, "example.py")

with open(script_uri, 'w') as f:
    print(script.strip(), file=f)

dataflow_configs = DataFlow(
    {
        "compartment_id": compartment_id,
        "driver_shape": driver_shape,
        "driver_shape_config": driver_shape_config,
        "executor_shape": executor_shape,
        "executor_shape_config": executor_shape_config,
        "logs_bucket_uri": log_bucket_uri,
        "metastore_id": metastore_id,
        "spark_version": spark_version
    }
)

runtime_config = DataFlowRuntime(
    {
        "script_uri": pyspark_file_path,
        "script_bucket": script_uri
    }
)

# creates a Data Flow application with DataFlow and DataFlowRuntime.
df_job = Job(name=job_name,
             infrastructure=dataflow_configs,
             runtime=runtime_config)
df_app = df_job.create()
df_run = df_app.run()

```

(continues on next page)

(continued from previous page)

```
# check a job log
df_run.watch()
```

### 13.5.2.2 Interactive Spark

```
from pyspark.sql import SparkSession

# Update these values
warehouse_uri = "<warehouse_uri>"
metastore_id = "<metastore_id>"

database_name = "ODSC_DEMO"
table_name = "ODSC_PYSPARK_METASTORE_DEMO"

# create a spark session
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL Hive integration example") \
    .config("spark.sql.warehouse.dir", warehouse_uri) \
    .config("spark.hadoop.oracle.dcat.metastore.id", metastore_id) \
    .enableHiveSupport() \
    .getOrCreate()
spark.sparkContext.setLogLevel("ERROR")

# show the databases in the warehouse:
spark.sql("SHOW DATABASES").show()
spark.sql(f"DROP DATABASE IF EXISTS {database_name} CASCADE")
spark.sql(f"CREATE DATABASE {database_name}")

# Load the Employee Attrition data file from OCI Object Storage into a Spark DataFrame:
file_path = "oci://hosted-ds-datasets@bigdatadatasciencelarge/synthetic/orcl_attrition.
↳ csv"
input_dataframe = spark.read.option("header", "true").csv(file_path)
input_dataframe.write.mode("overwrite").saveAsTable(f"{database_name}.{table_name}")

# explore data
spark_df = spark.sql(f"""
    SELECT EducationField, SalaryLevel, JobRole FROM {database_name}.
↳ {table_name} limit 10
    """)
spark_df.show()
```

### 13.5.3 Data Flow

This example demonstrates how to create a Data Flow application that is connected to the Data Catalog Metastore. It creates a PySpark script, then a Data Flow application. This application can be run by directly by Data Flow or as part of a Job.

This section runs Hive queries using Data Flow. When the Data Catalog is being used the only changes that need to be made are to provide the metastore OCID.

#### 13.5.3.1 PySpark Script

A PySpark script is needed for the Data Flow application. The following code creates that script. The script will use Spark to load a CSV file from a public Object Storage bucket. It will then create a database and write the file to Object Storage. Finally, it will use Spark SQL to query the database and print the records in JSON format.

There is nothing in the PySpark script that is specific to using Data Catalog Metastore. The script treats the database as a standard Hive database.

```
script = '''
from pyspark.sql import SparkSession

def main():

    database_name = "employee_attrition"
    table_name = "orcl_attrition"

    # Create a Spark session
    spark = SparkSession \\\
        .builder \\\
        .appName("Python Spark SQL basic example") \\\
        .enableHiveSupport() \\\
        .getOrCreate()

    # Load a CSV file from a public Object Storage bucket
    df = spark \\\
        .read \\\
        .format("csv") \\\
        .option("header", "true") \\\
        .option("multiLine", "true") \\\
        .load("oci://hosted-ds-datasets@bigdatadatasciencelarge/synthetic/orcl_attrition.
↪ csv")

    print(f"Creating {database_name}")
    spark.sql(f"DROP DATABASE IF EXISTS {database_name} CASCADE")
    spark.sql(f"CREATE DATABASE IF NOT EXISTS {database_name}")

    # Write the data to the database
    df.write.mode("overwrite").saveAsTable(f"{database_name}.{table_name}")

    # Use Spark SQL to read from the database.
    query_result_df = spark.sql(f"""
        SELECT EducationField, SalaryLevel, JobRole FROM
↪ {database_name}.{table_name} limit 10
        """)
```

(continues on next page)

(continued from previous page)

```

    # Convert the filtered Apache Spark DataFrame into JSON format and write it out to
    ↪ stdout
    # so that it can be captured in the log.
    print('\n'.join(query_result_df.toJSON().collect()))

if __name__ == '__main__':
    main()
'''

# Save the PySpark script to a file
dataflow_base_folder = tempfile.mkdtemp()
script_uri = os.path.join(dataflow_base_folder, "example.py")

with open(script_uri, 'w') as f:
    print(script.strip(), file=f)

```

### 13.5.3.2 Create Application

To create a Data Flow application you will need `DataFlow` and `DataFlowRuntime` objects. A `DataFlow` object stores the properties that are specific to the Data Flow service. These would be things such as the compartment OCID, the URI to the Object Storage bucket for the logs, the type of hardware to be used, the version of Spark, and much more. If you are using a Data Catalog Metastore to manage a database, the metastore OCID is stored in this object. The `DataFlowRuntime` object stores properties related to the script to be run. This would be the bucket to be used for the script, the location of the PySpark script, and any command-line arguments.

Update the `script_bucket`, `log_bucket`, and `metastore_id` variables to match your tenancy's configuration.

```

# Update values
log_bucket_uri = "oci://<bucket_name>@<namespace>/<prefix>"
metastore_id = "<metastore_id>"
script_bucket = "oci://<bucket_name>@<namespace>/<prefix>"

compartment_id = os.environ.get("NB_SESSION_COMPARTMENT_OCID")
driver_shape = "VM.Standard.E4.Flex"
driver_shape_config = {"ocpus":2, "memory_in_gbs":32}
executor_shape = "VM.Standard.E4.Flex"
executor_shape_config = {"ocpus":4, "memory_in_gbs":64}
spark_version = "3.2.1"

```

In the following example, a `DataFlow` is created and populated with the information that it needs to define the Data Flow service. Since, we are connecting to the Data Catalog Metastore to work with a Hive database, the metastore OCID must be given.

```

from ads.jobs import DataFlow, DataFlowRun, DataFlowRuntime

dataflow_configs = DataFlow(
    {"compartment_id": compartment_id,
     "driver_shape": driver_shape,
     "driver_shape_config": driver_shape_config,
     "executor_shape": executor_shape,
     "executor_shape_config": executor_shape_config,

```

(continues on next page)

(continued from previous page)

```

    "logs_bucket_uri": log_bucket_uri,
    "metastore_id": metastore_id,
    "spark_version": spark_version}
)

```

In the following example, a `DataFlowRuntime` is created and populated with the URI to the PySpark script and the URI for the script bucket. The script URI specifies the path to the script. It can be local or remote (an Object Storage path). If the path is local, then a URI to the script bucket must also be specified. This is because Data Flow requires a script to be in Object Storage. If the specified path to the PySpark script is on a local drive, ADS will upload it for you.

```

runtime_config = DataFlowRuntime(
    {
        "script_bucket": script_uri
        "script_uri": pyspark_file_path,
    }
)

```

### 13.5.3.3 Run

The recommended approach for running Data Flow applications is to use a Job. This will prevent your notebook from being blocked.

A Job requires a name, infrastructure, and runtime settings. Update the following code to give the job a unique name. The `infrastructure` takes a `DataFlow` object and the `runtime` parameter takes a `DataFlowRuntime` object.

```

# Update values
job_name = "<job_name>"

df_job = Job(name=job_name,
             infrastructure=dataflow_configs,
             runtime=runtime_config)
df_app = df_job.create()
df_run = df_app.run()

```

## 13.5.4 Interactive Spark

This section demonstrates how to make connections to the Data Catalog Metastore and Object Storage. It uses Spark to load data from a public Object Storage file and creates a database. The metadata for the database is managed by the Data Catalog Metastore and the data is copied to your data warehouse bucket. Finally, Spark is used to make a Spark SQL query on the database.

Specify the bucket URI that will act as the data warehouse. Use the `warehouse_uri` variable and it should have the following format `oci://<bucket_name>@<namespace_name>/<prefix>`. Update the variable `metastore_id` with the OCID of the Data Catalog Metastore.

Create a Spark session that connects to the Data Catalog Metastore and the Object Storage that will act as the data warehouse.

```

from pyspark.sql import SparkSession

warehouse_uri = "<warehouse_uri>"

```

(continues on next page)



(continued from previous page)

```

metastore_id = "<metastore_id>"

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL Hive integration example") \
    .config("spark.sql.warehouse.dir", warehouse_uri) \
    .config("spark.hadoop.oracle.dcat.metastore.id", metastore_id) \
    .enableHiveSupport() \
    .getOrCreate()
spark.sparkContext.setLogLevel("ERROR")

```

Load a data file from Object Storage into a Spark DataFrame. Create a database in the Data Catalog Metastore and then save the dataframe as a table. This will write the files to the location specified by the `warehouse_uri` variable.

```

database_name = "ODSC_DEMO"
table_name = "ODSC_PYSPARK_METASTORE_DEMO"
file_path = "oci://hosted-ds-datasets@bigdatadatasciencelarge/synthetic/orcl_attrition.
↳ csv"

input_dataframe = spark.read.option("header", "true").csv(file_path)
spark.sql(f"DROP DATABASE IF EXISTS {database_name} CASCADE")
spark.sql(f"CREATE DATABASE {database_name}")
input_dataframe.write.mode("overwrite").saveAsTable(f"{database_name}.{table_name}")

```

Use Spark SQL to read from the database.

```

spark_df = spark.sql(f"""
                        SELECT EducationField, SalaryLevel, JobRole FROM {database_name}.
↳ {table_name} limit 10
                        """)
spark_df.show()

```

## 13.6 OCI Data Flow Studio

This section demonstrates how to run interactive Spark workloads on a long lasting [Oracle Cloud Infrastructure Data Flow](#) cluster through [Apache Livy](#) integration.

**Data Flow Studio allows you to:**

- Run Spark code against a Data Flow remote Spark cluster
- Create a Data Flow Spark session with `SparkContext` and `HiveContext` against a Data Flow remote Spark cluster
- Capture the output of Spark queries as a local Pandas data frame to interact easily with other Python libraries (e.g. `matplotlib`)

**Key Features & Benefits:**

- Data Flow sessions support auto-scaling Data Flow cluster capabilities
- Data Flow sessions support the use of conda environments as customizable Spark runtime environments

**Limitations:**

- Data Flow sessions can last up to 7 days or 10,080 mins (`maxDurationInMinutes`).

- Data Flow Sessions can only be accessed through OCI Data Science Notebook Sessions.
- Not all SparkMagic commands are currently supported. To see the full list, run the `%help` command in a notebook cell.

#### Notebook Examples:

- [Introduction to the Oracle Cloud Infrastructure Data Flow Studio](#)
- [Spark NLP within Oracle Cloud Infrastructure Data Flow Studio](#)

## 13.6.1 Prerequisite

Data Flow Sessions are accessible through the following conda environment:

- PySpark 3.2 and Data Flow 2.0 (pyspark32\_p38\_cpu\_v2)

You can customize `pyspark32_p38_cpu_v1`, publish it, and use it as a runtime environment for a Data Flow Session.

### 13.6.1.1 Policies

Data Flow requires policies to be set in IAM to access resources to manage and run sessions. Refer to the [Data Flow Studio Policies](#) documentation on how to setup policies.

## 13.6.2 Quick Start

```
import ads
ads.set_auth("resource_principal")

%load_ext dataflow.magics

%create_session -l python -c '{\
  "compartmentId": "<compartment_id>",\
  "displayName": "TestDataFlowSession",\
  "sparkVersion": "3.2.1",\
  "driverShape": "VM.Standard.E4.Flex",\
  "executorShape": "VM.Standard.E4.Flex",\
  "numExecutors": 1,\
  "driverShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "executorShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "logsBucketUri" : "oci://<bucket_name>@<namespace>/'

%%spark
print(sc.version)
```

### 13.6.3 Data Flow Spark Magic

Data Flow Spark Magic is used for interactively working with remote Spark clusters through Livy, a Spark REST server, in Jupyter notebooks. It is a JupyterLab extension that you need to activate in your notebook.

```
%load_ext dataflow.magics
```

Use the `%help` method to get a list of all the available commands, along with a list of their arguments and example calls.

```
%help
```

#### Tip

To access the docstrings of any magic command and figure out what arguments to provide, simply add `?` at the end of the command. For instance: `%create_session?`

#### 13.6.3.1 Create Session

##### Example command for Flex shapes

To create a new Data Flow cluster session use the `%create_session` magic command.

```
%create_session -l python -c '{\
  "compartmentId": "<compartment_id>",\
  "displayName": "TestDataFlowSession",\
  "sparkVersion": "3.2.1",\
  "driverShape": "VM.Standard.E4.Flex",\
  "executorShape": "VM.Standard.E4.Flex",\
  "numExecutors": 1,\
  "driverShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "executorShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "logsBucketUri" : "oci://<bucket_name>@<namespace>/"}'
```

##### Example command for Spark dynamic allocation (aka auto-scaling)

To help you save resources and reduce time on management, Spark [dynamic allocation](#) is now enabled in Data Flow. You can define a Data Flow cluster based on a range of executors, instead of just a fixed number of executors. Spark provides a mechanism to dynamically adjust the resources the application occupies based on the workload. The application might relinquish resources if they are no longer used and request them again later when there is demand.

```
%create_session -l python -c '{\
  "compartmentId": "<compartment_id>",\
  "displayName": "TestDataFlowSession",\
  "sparkVersion": "3.2.1",\
  "driverShape": "VM.Standard.E4.Flex",\
  "executorShape": "VM.Standard.E4.Flex",\
  "numExecutors": 1,\
  "driverShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "executorShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "logsBucketUri" : "oci://<bucket_name>@<namespace>/"\
  "configuration": {\
    "spark.dynamicAllocation.enabled": "true",\
```

(continues on next page)

(continued from previous page)

```
"spark.dynamicAllocation.shuffleTracking.enabled":"true",\
"spark.dynamicAllocation.minExecutors":"1",\
"spark.dynamicAllocation.maxExecutors":"4",\
"spark.dynamicAllocation.executorIdleTimeout":"60",\
"spark.dynamicAllocation.schedulerBacklogTimeout":"60",\
"spark.dataflow.dynamicAllocation.quotaPolicy":"min"}}'
```

### Example command with third-party libraries

The Data Flow Sessions support [custom dependencies](#) in the form of Python wheels or virtual environments. You might want to make native code or other assets available within your Spark runtime. The dependencies can be attached by using the `archiveUri` attribute.

```
%create_session -l python -c '{\
  "compartmentId":"<compartment_id>",\
  "displayName":"TestDataFlowSession",\
  "sparkVersion":"3.2.1",\
  "driverShape":"VM.Standard.E4.Flex",\
  "executorShape":"VM.Standard.E4.Flex",\
  "numExecutors":1,\
  "driverShapeConfig":{"ocpus":1,"memoryInGBs":16},\
  "executorShapeConfig":{"ocpus":1,"memoryInGBs":16},\
  "archiveUri":"oci://<bucket_name>@<namespace>/<zip_archive>",\
  "logsBucketUri" : "oci://<bucket_name>@<namespace>/"}'
```

### Example command with the Data Catalog Hive Metastore

The [Data Catalog Hive Metastore](#) provides schema definitions for objects in structured and unstructured data assets. Use the `metastoreId` to access the Data Catalog Metastore.

```
%create_session -l python -c '{\
  "compartmentId":"<compartment_id>",\
  "displayName":"TestDataFlowSession",\
  "sparkVersion":"3.2.1",\
  "driverShape":"VM.Standard.E4.Flex",\
  "executorShape":"VM.Standard.E4.Flex",\
  "numExecutors":1,\
  "driverShapeConfig":{"ocpus":1,"memoryInGBs":16},\
  "executorShapeConfig":{"ocpus":1,"memoryInGBs":16},\
  "metastoreId": "<ocid1.datacatalogmetastore...>",\
  "logsBucketUri" : "oci://<bucket_name>@<namespace>/"}'
```

### Example command with the published conda environment

You can use a published conda environment as a Data Flow runtime environment.

- [Creating a Custom Conda Environment](#)
- [How to create a new conda environment in OCI Data Science](#)
- [Publishing a Conda Environment to an Object Storage Bucket in Your Tenancy](#)

The path to the published conda environment can be copied from the [Environment Explorer](#).

Example path : `oci://<your-bucket>@<your-tenancy-namespace>/conda_environments/cpu/PySpark 3.2 and Data Flow/2.0/pyspark32_p38_cpu_v2#conda`

```
%create_session -l python -c '{\
  "compartmentId": "<compartment_id>",\
  "displayName": "TestDataFlowSession",\
  "sparkVersion": "3.2.1",\
  "driverShape": "VM.Standard.E4.Flex",\
  "executorShape": "VM.Standard.E4.Flex",\
  "numExecutors": 1,\
  "driverShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "executorShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "logsBucketUri" : "oci://<bucket_name>@<namespace>/"\
  "configuration": {\
    "spark.archives": "oci://<your-bucket>@<your-tenancy-namespace>/conda_environments/\
    ↪cpu/PySpark 3.2 and Data Flow/2.0/pyspark32_p38_cpu_v2#conda"} }'
```

### 13.6.3.2 Update Session

You can modify the configuration of your running session using the `%update_session` command. For example, Data Flow sessions can last up to 7 days or 10080 mins (168 hours) (**maxDurationInMinutes**) and have default idle timeout value of 480 mins (8 hours)(**idleTimeoutInMinutes**). Only those two can be updated on a running cluster without re-creating the cluster.

```
%update_session -i '{"maxDurationInMinutes": 1440, "idleTimeoutInMinutes": 420}'
```

### 13.6.3.3 Configure Session

The existing session can be reconfigured with the `%configure_session` command. The new configuration will be applied the next time the session is started. Use the force flag `-f` to immediately drop and recreate the running cluster session.

```
%configure_session -f -i '{\
  "driverShape": "VM.Standard.E4.Flex",\
  "executorShape": "VM.Standard.E4.Flex",\
  "numExecutors": 2,\
  "driverShapeConfig": {"ocpus": 1, "memoryInGBs": 16},\
  "executorShapeConfig": {"ocpus": 1, "memoryInGBs": 16}}'
```

### 13.6.3.4 Stop Session

To stop the current session, use the `%stop_session` magic command. You don't need to provide any arguments for this command. The current active cluster will be stopped. All data in memory will be lost.

```
%stop_session
```

### 13.6.3.5 Activate Session

To re-activate the existing session, use the `%activate_session` magic command. The `application_id` can be taken from the console UI.

```
%activate_session -l python -c '{\n  "compartmentId": "<compartment_id>",\n  "displayName": "TestDataFlowSession",\n  "applicationId": "<application_id>"}'
```

### 13.6.3.6 Use Existing Session

To connect to the existing session use the `%use_session` magic command.

```
%use_session -s <application_id>
```

## 13.6.4 Basic Spark Usage Examples

A `SparkContext` (`sc`) and `HiveContext` (`sqlContext`) are automatically created in the session cluster. The magic commands include the `%%spark` command to run Spark commands in the cluster. You can access information about the Spark application, define a dataframe where results are to be stored, modify the configuration, and so on.

The `%%spark` magic command comes with a number of parameters that allow you to interact with the Data Flow Spark cluster. Any cell content that starts with the `%%spark` command will be executed in the remote Spark cluster.

Check the Spark context version:

```
%%spark\nprint(sc.version)
```

A toy example of how to use `sc` in a Data Flow Spark Magic cell:

```
%%spark\nnumbers = sc.parallelize([4, 3, 2, 1])\nprint(f"First element of numbers is {numbers.first()}")\nprint(f"The RDD, numbers, has the following description\\n{numbers.toDebugString()}")
```

### 13.6.4.1 Spark SQL

Using the `-c sql` option allows you to run Spark SQL commands in a cell. In this section, the [NYC Taxi and Limousine Commission \(TLC\) Data](#) dataset is used. The size of the dataset is around **35GB**.

The next cell reads the dataset into a Spark dataframe, and then saves it as a view used to demonstrate Spark SQL.

Use the `-c sql` option to run Spark SQL commands in a cell.

The next example demonstrates how a dataset can be created on the fly:

```
%%spark\ndf_nyc_tlc = spark.read.parquet("oci://hosted-ds-datasets@bigdatadatasciencelarge/nyc_↵\ntlc/201[1,2,3,4,5,6,7,8]**/data.parquet", header=False, inferSchema=True)\ndf_nyc_tlc.show()\ndf_nyc_tlc.createOrReplaceTempView("nyc_tlc")
```

The following cell uses the `-c sql` option to tell Data Flow Spark Magic that the contents of the cell is SparkSQL. The `-o <variable>` option takes the results of the Spark SQL operation and stores it in the defined variable. In this case, the `df_people` will be a Pandas dataframe that is available to be used in the notebook.

```
%%spark -c sql -o df_nyc_tlc
SELECT vendor_id, passenger_count, trip_distance, payment_type FROM nyc_tlc LIMIT 1000;
```

Check the result:

```
print(type(df_nyc_tlc))
df_nyc_tlc.head()
```

## 13.7 [Legacy]

Deprecated since version v2.6.3: July 2022

ADS can be used to create and run PySpark applications directly from a notebook session.

### 13.7.1 Prerequisite

To access , there are a number of steps that are needed to be completed.

- DataFlow requires a bucket to store the logs, and a data warehouse bucket. Refer to the Data Flow documentation for [setting up storage](#).
- DataFlow requires policies to be set in IAM to access resources to manage and run applications/sessions. Refer to the Data Flow documentation on how to [setup policies](#).
- DataFlow natively supports conda packs published to OCI Object Storage. Ensure the Data Flow Resource has read access to the bucket or path of your published conda pack, and that the spark version  $\geq 3$  when running your Data Flow Application/Session.
- The `core-site.xml` file needs to be configured.

### 13.7.2 Create a Instance

First, you create a DataFlow object instance.

By default, all artifacts are stored using the `dataflow_base_folder` optional argument. By default, all artifacts are stored in `/home/datascience/dataflow`. The `dataflow_base_folder` directory contains multiple subdirectories, each one corresponds to a different application. The name of the subdirectory corresponds to the application name that a random string is added as a suffix. In each application directory, artifacts generated by separate runs are stored in different folders. Each folder is identified by the run display name and the run creation time. All the run specific artifacts including the script, the run configuration, and the run logs are saved in the corresponding run folder.

Also, you can choose to use a specific compartment using the optional `compartment_id` argument when creating the dataflow instance. Otherwise, it uses the **same** compartment as **your notebook session** to create the instance.

```
from ads.dataflow.dataflow import DataFlow
data_flow = DataFlow(
    compartment_id="<compartmentA_OCID>",
    dataflow_base_folder="<my_dataflow_dir>"
)
```

### 13.7.3 Generate a Script Using a Template

We provide simple PySpark or sparksql templates for you to get started with . You can use `data_flow.template()` to generate a pre-written template.

We support these templates:

The `standard_pyspark` template is used for standard PySpark jobs.

The `sparksql` template is used for sparksql jobs.

```
from ads.dataflow.dataflow import DataFlow
data_flow = DataFlow()
data_flow.template(job_type='standard_pyspark')
```

`data_flow.template()` returns the local path to the script you have generated.

### 13.7.4 Create a Application

The application creation process has two stages, preparation and creation.

In the preparation stage, you prepare the configuration object necessary to create a application. You must provide values for these three parameters:

- `display_name`: The name you give your application.
- `pyspark_file_path`: The local path to your PySpark script.
- `script_bucket`: The bucket used to read/write the PySpark script in Object Storage.

ADS checks that the bucket exists, and that you can write to it from your notebook session. Optionally, you can change values for these parameters:

- `compartment_id`: The OCID of the compartment to create a application. If it's not provided, the same compartment as your dataflow object is used.
- `driver_shape`: The driver shape used to create the application. The default value is "VM.Standard2.4".
- `executor_shape`: The executor shape to create the application. The default value is "VM.Standard2.4".
- `logs_bucket`: The bucket used to store run logs in Object Storage. The default value is "dataflow-logs".
- `num_executors`: The number of executor VMs requested. The default value is 1.

---

**Note:** If you want to use a private bucket as the `logs_bucket`, ensure that you add a corresponding service policy using `Identity: Policy Set Up` <[https://docs.cloud.oracle.com/en-us/iaas/data-flow/using/dfs\\_getting\\_started.htm#policy\\_set\\_up](https://docs.cloud.oracle.com/en-us/iaas/data-flow/using/dfs_getting_started.htm#policy_set_up)>`.

---

Then you can use `prepare_app()` to create the configuration object necessary to create the application.

```
from ads.dataflow.dataflow import DataFlow

data_flow = DataFlow()
app_config = data_flow.prepare_app(
    display_name="<app-display-name>",
    script_bucket="<your-script-bucket>" ,
    pyspark_file_path="<your-script-path>"
)
```



After you have the application configured, you can create an application using `create_app`:

```
app = data_flow.create_app(app_config)
```

Your local script is uploaded to the script bucket in this application creation step. Object Storage supports file versioning that creates an object version when the content changes, or the object is deleted. You can enable **Object Versioning** in your bucket in the OCI Console to prevent overwriting of existing files in Object Storage.

You can create an application with a script file that exists in Object Storage by setting `overwrite_script=True` in `create_app`. Similarly, you can set `overwrite_archive=True` to create an application with an archive file that exists in Object Storage. By default, the `overwrite_script` and `overwrite_archive` options are set to `false`.

```
app = data_flow.create_app(app_config, overwrite_script=True, overwrite_archive=True)
```

You can explore a few attributes of the `DataFlowApp` object.

First, you can look at the configuration of the application.

```
app.config
```

Next, you could get a URL link to the OCI Console Application Details page.

```
app.oci_link
```

### 13.7.4.1 Load an Existing Application

As an alternative to creating applications in ADS, you can load existing applications created elsewhere. These applications must be Python applications. To load an existing applications, you need the application's OCID.

```
existing_app = data_flow.load_app(app_id, target_folder)
```

You can find the `app_id` in the the OCI Console or by listing existing applications.

Optionally, you could assign a value to the parameter `target_folder`. This parameter is the directory you want to store the local artifacts of this application in. If `target_folder` is not provided, then the local artifacts of this application are stored in the `dataflow_base_folder` folder defined by the `dataflow` object instance.

### 13.7.4.2 Listing Applications

From ADS you can list applications, that are returned as a list of dictionaries, with a function to provide the data in a Pandas dataframe. The default sort order is the most recent run first.

For example, to list the most recent five applications use this code:

```
from ads.dataflow.dataflow import DataFlow
data_flow = DataFlow()
data_flow.list_apps().to_dataframe().head(5)
```

### 13.7.4.3 Create a Run

After an application is created or loaded in your notebook session, the next logical step is to execute a run of that application. The process of running (or creating) a run is similar to creating an application.

First, you configure the run using the `prepare_run()` method of the `DataFlowApp` object. You only need to provide a value for the name of your run using `run_display_name`:

```
run_config = app.prepare_run(run_display_name="<run-display-name>")
```

You could use a compartment different from your application to create a run by specifying the `compartment_id` in `prepare_run`. By default, it uses the same compartment as your application to create the run.

Optionally, you can specify the `logs_bucket` to store the logs of your run. By default, the run inherits the `logs_bucket` from the parent application, but you can overwrite that option.

Every time the application launches a run, a local folder representing this run is created. This folder stores all the information including the script, the run configuration, and any logs that are stored in the logs bucket.

Then, you can create a run using the `run_config` generated in the preparation stage. During this process, you can monitor the run while the job is running. You can also pull logs into your local directories by setting, `save_log_to_local=True`.

```
run = app.run(run_config, save_log_to_local=True)
```

The `DataFlowRun` object has some useful attributes similar to the `DataFlowApp` object.

You can check the status of the run with:

```
run.status
```

You can get the configuration file that created this run. The run configuration and the PySpark script used in this run are also saved in the corresponding run directory in your notebook environment.

```
run.config
```

You can get the run directory where the artifacts are stored in your notebook environment with:

```
run.local_dir
```

Similarly, you can get a clickable link to the OCI Console Run Details page with:

```
run.oci_link
```

### 13.7.4.4 Fetching Logs

After a run has completed, you can examine the logs using ADS. There are two types of logs, `stdout` and `stderr`.

```
run.log_stdout.head()  # show first rows of stdout
run.log_stdout.tail()  # show last lines of stdout
```

```
# where the logs are stored on OCI Storage
```

```
run.log_stdout.oci_path
```

```
# the path to the saved logs in the notebook environment if ``save_log_to_local`` was_
```

(continues on next page)

(continued from previous page)

```
→ ``True`` when you create this run
run.log_stdout.local_path
```

If `save_log_to_local` is set to `False` during `app.run(...)`, you can fetch logs by calling the `fetch_log(...).save()` method on the `DataFlowRun` object with the correct logs type.

```
run.fetch_log("stdout").save()
run.fetch_log("stderr").save()
```

**Note:** Due to a limitation of PySpark (specifically Python applications in Spark), both `stdout` and `stderr` are merged into the `stdout` stream.

#### 13.7.4.5 Edit and Synchronize PySpark Script

The integration with ADS supports the edit-run-edit cycle, so the local PySpark script can be edited, and is automatically synchronized to Object Storage each time the application is run.

obtains the PySpark script from Object Storage

so the local files in the notebook session are not visible to . The `app.run(...)` method compares the content hash of the local file with the remote copy on Object Storage. If any change is detected, the new local version is copied over to the remote. For the first run the synchronization creates the remote file and generates a fully qualified URL with namespace that's required for .

Synchronizing is the default setting in `app.run(...)`. If you don't want the application to sync with the local modified files, you need to include `sync=False` as an argument parameter in `app.run(...)`.

#### 13.7.4.6 Arguments and Parameters

Passing arguments to PySpark scripts is done with the `arguments` value in `prepare_app`. Additional to the arguments supports, is a parameter dictionary that you can use to interpolate arguments. To just pass arguments, the `script_parameter` section may be ignored. However, any key-value pair defined in `script_parameter` can be referenced in arguments using the `${key}` syntax, and the value of that key is passed as the argument value.

```
from ads.dataflow.dataflow import DataFlow

data_flow = DataFlow()
app_config = data_flow.prepare_app(
    display_name,
    script_bucket,
    pyspark_file_path,
    arguments = ['${foo}', 'bar', '-d', '--file', '${filename}'],
    script_parameters={
        'foo': 'val1 val2',
        'filename': 'file1',
    }
)
app = data_flow.create_app(app_config)

run_config = app.prepare_run(run_display_name="test-run")
run = app.run(run_config)
```

**Note:** The arguments in the format of `${arg}` are replaced by the value provided in script parameters when passed in, while arguments not in this format are passed into the script verbatim.

---

You can override the values of some or all script parameters in each run by passing different values to `prepare_run()`.

```
run_config = app.prepare_run(run_display_name="test-run", foo='val3')
run = app.run(run_config)
```

#### 13.7.4.7 Add Third-Party Libraries

Your PySpark applications might have custom dependencies in the form of Python wheels or virtual environments, see [Adding Third-Party Libraries to Applications](#).

Pass the archive file to your applications with `archive_path` and `archive_bucket` values in `prepare_app`.

- `archive_path`: The local path to archive file.
- `archive_bucket`: The bucket used to read and write the archive file in Object Storage; if not provided, `archive_bucket` will use the bucket for PySpark bucket by default.

Use `prepare_app()` to create the configuration object necessary to create the application.

```
from ads.dataflow.dataflow import DataFlow

data_flow = DataFlow()
app_config = data_flow.prepare_app(
    display_name="<app-display-name>",
    script_bucket="<your-script-bucket>",
    pyspark_file_path="<your-script-path>",
    archive_path="<your-archive-path>",
    archive_bucket="<your-archive-bucket>"
)
```

The behavior of the archive file is very similar to the PySpark script when creating:

- An application, the local archive file is uploaded to the specified bucket Object Storage.
- A run, the latest local archive file is synchronized to the remote file in Object Storage. The `sync` parameter controls this behavior.
- Loading an existing application created with `archive_uri`, the archive file is obtained from Object Storage, and saved in the local directory.

#### 13.7.4.8 Fetching PySpark Output

After the application has run and any `stdout` captured in the log file, the PySpark script likely produces some form of output. Usually a PySpark script batch processes something. For example, sampling data, aggregating data, preprocessing data. You can load the resulting output as an `ADSDataset.open()` using the `ocis://` protocol handler.

The only way to get output from PySpark back into the notebook session is to create files in Object Storage that is read into the notebook, or use the `stdout` stream.

Following is a simple example of a PySpark script producing output printed in a portable JSON-L format, though CSV works too. This method, while convenient as an example, is not a recommended for large data.

```

from pyspark.sql import SparkSession

def main():

    # create a spark session
    spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .getOrCreate()

    # load an example csv file from dataflow public storage into DataFrame
    original_df = spark \
        .read \
        .format("csv") \
        .option("header", "true") \
        .option("multiLine", "true") \
        .load("oci://oow_2019_dataflow_lab@bigdatadatasciencelarge/usercontent/kaggle_
↳berlin_airbnb_listings_summary.csv")

    # the dataframe as a sql view so we can perform SQL on it
    original_df.createOrReplaceTempView("berlin")

    query_result_df = spark.sql("""
        SELECT
            city,
            zipcode,
            number_of_reviews,
            CONCAT(latitude, ',', longitude) AS lat_long
        FROM
            berlin""")

    )

    # Convert the filtered Spark DataFrame into JSON format
    # Note: we are writing to the spark stdout log so that we can retrieve the log later.
↳at the end of the notebook.

    print('\n' \
        .join(query_result_df \
            .toJSON() \
            .collect()))

if __name__ == '__main__':
    main()

```

After you run the stdout stream (which contains CSV formatted data), it can be interpreted as a string using Pandas.

```

import io
import pandas as pd

# the PySpark script wrote to the log as jsonL, and we read the log back as a pandas
↳dataframe
df = pd.read_json((str(run.log_stdout)), lines=True)

```

(continues on next page)

(continued from previous page)

```
df.head()
```

### 13.7.5 Example Notebook: Develop Pyspark jobs locally - from local to remote workflows

This notebook provides spark operations for customers by bridging the existing local spark workflows with cloud based capabilities. Data scientists can use their familiar local environments with JupyterLab, and work with remote data and remote clusters simply by selecting a kernel. The operations demonstrated are, how to:

- Use the interactive spark environment and produce a spark script,
- Prepare and create an application,
- Prepare and create a run,
- List existing dataflow applications,
- Retrieve and display the logs,

The purpose of the dataflow module is to provide an efficient and convenient way for you to launch a Spark application, and run Spark jobs. The interactive Spark kernel provides a simple and efficient way to edit and build your Spark script, and easy access to read from an OCI filesystem.

```
import io
import matplotlib.pyplot as plt
import os
from os import path
import pandas as pd
import tempfile
import uuid

from ads.dataflow.dataflow import DataFlow

from pyspark.sql import SparkSession
```

#### Build your PySpark Script Using an Interactive Spark kernel

Set up spark session in your PySpark conda environment:

```
# create a spark session
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.driver.cores", "4") \
    .config("spark.executor.cores", "4") \
    .getOrCreate()
```

Load the Employee Attrition data file from OCI Object Storage into a Spark DataFrame:

```
emp_attrition = spark\
    .read\
    .format("csv")\
    .option("header", "true")\
    .option("inferSchema", "true")\
```

(continues on next page)

(continued from previous page)

```

.option("multiline", "true")\
.load("oci://hosted-ds-datasets@bigdatadatasciencelarge/synthetic/orcl_attrition.
↪ csv") \
.cache() # cache the dataset to increase computing speed
emp_attrition.createOrReplaceTempView("emp_attrition")

```

Next, explore the dataframe:

```
spark.sql('select * from emp_attrition limit 5').toPandas()
```

Visualize how monthly income and age relate to one another in the context of years in industry:

```

fig, ax = plt.subplots()
plot = spark.sql("""
    SELECT
        Age,
        MonthlyIncome,
        YearsInIndustry
    FROM
        emp_attrition
    """).toPandas().plot.scatter(x="Age", y="MonthlyIncome", title='Age vs Monthly_
↪ Income',
                                c="YearsInIndustry", cmap="viridis", figsize=(12,
↪ 12), ax=ax)
plot.set_xlabel("Age")
plot.set_ylabel("Monthly Income")
plot

```

```

<AxesSubplot:title={'center':'Age vs Monthly Income'}, xlabel='Age', ylabel='Monthly_
↪ Income'>

```

View all of the columns in the table:

```
spark.sql("show columns from emp_attrition").show()
```

```

+-----+
|      col_name|
+-----+
|      Age|
|  Attrition|
| TravelForWork|
|  SalaryLevel|
|  JobFunction|
| CommuteLength|
| EducationalLevel|
| EducationField|
|      Directs|
| EmployeeNumber|
| EnvironmentSatisf..|
|      Gender|
|   HourlyRate|
| JobInvolvement|

```

(continues on next page)

(continued from previous page)

```
|          JobLevel|
|          JobRole|
|    JobSatisfaction|
|      MaritalStatus|
|      MonthlyIncome|
|      MonthlyRate|
+-----+
only showing top 20 rows
```

Select a few columns using Spark, and convert it into a Pandas dataframe:

```
df = spark.sql("""
    SELECT
        Age,
        MonthlyIncome,
        YearsInIndustry
    FROM
        emp_attrition """).limit(10).toPandas()

df
```

You can work with different compression formats within . For example, snappy Parquet:

```
# Writing to a snappy parquet file
df.to_parquet('emp_attrition.parquet.snappy', compression='snappy')
pd.read_parquet('emp_attrition.parquet.snappy')
```

```
# We are able to read in this snappy parquet file to a spark dataframe
read_snappy_df = SparkSession \
    .builder \
    .appName("Snappy Compression Loading Example") \
    .config("spark.io.compression.codec", "org.apache.spark.io.SnappyCompressionCodec") \
    .getOrCreate() \
    .read \
    .format("parquet") \
    .load(f"{os.getcwd()}/emp_attrition.parquet.snappy")

read_snappy_df.first()
```

```
Row(Age=42, MonthlyIncome=5993, YearsInIndustry=8)
```

Other compression formats that supports include snappy Parquet, and Gzip on both CSV and Parquet.

You might have query that you want to run in from previous explorations, review the *dataflow.ipynb* notebook example that shows you how to submit a job to .

```
dataflow_base_folder = tempfile.mkdtemp()
data_flow = DataFlow(dataflow_base_folder=dataflow_base_folder)
print("Data flow directory: {}".format(dataflow_base_folder))
```

```
Data flow directory: /tmp/tmpel18x_qbr
```



```

pyspark_file_path = path.join(dataflow_base_folder, "example-{}.py".format(str(uuid.
↳ uuid4())[-6:]))
script = '''
from pyspark.sql import SparkSession

def main():

    # Create a Spark session
    spark = SparkSession \\\
        .builder \\\
        .appName("Python Spark SQL basic example") \\\
        .getOrCreate()

    # Load a csv file from dataflow public storage
    df = spark \\\
        .read \\\
        .format("csv") \\\
        .option("header", "true") \\\
        .option("multiLine", "true") \\\
        .load("oci://hosted-ds-datasets@bigdatadatasciencelarge/synthetic/orcl_attrition.
↳ csv")

    # Create a temp view and do some SQL operations
    df.createOrReplaceTempView("emp_attrition")
    query_result_df = spark.sql("""
        SELECT
            Age,
            MonthlyIncome,
            YearsInIndustry
        FROM emp_attrition
    """)

    # Convert the filtered Spark DataFrame into JSON format
    # Note: we are writing to the spark stdout log so that we can retrieve the log later.
↳ at the end of the notebook.
    print('\n'.join(query_result_df.toJSON().collect()))

if __name__ == '__main__':
    main()
'''

with open(pyspark_file_path, 'w') as f:
    print(script.strip(), file=f)

print("Script path: {}".format(pyspark_file_path))

```

```
Script path: /tmp/example.py
```

```

script_bucket = "test"                # Update the value
logs_bucket = "dataflow-log"          # Update the value
display_name = "sample_Data_Flow_app"

```

(continues on next page)

(continued from previous page)

```

app_config = data_flow.prepare_app(display_name=display_name,
                                   script_bucket=script_bucket,
                                   pyspark_file_path=pyspark_file_path,
                                   logs_bucket=logs_bucket)

app = data_flow.create_app(app_config)

run_display_name = "sample_Data_Flow_run"
run_config = app.prepare_run(run_display_name=run_display_name)

run = app.run(run_config, save_log_to_local=True)

```

```
run.status
```

```
'SUCCEEDED'
```

```
run.config
```

```

{'compartment_id': 'ocid1.compartment.<unique_ID>',
 'script_bucket': 'test',
 'pyspark_file_path': '/tmp/tmpel8x_qbr/example-0054ed.py',
 'archive_path': None,
 'archive_bucket': None,
 'run_display_name': 'sample_Data_Flow_run',
 'logs_bucket': 'dataflow-log',
 'logs_bucket_uri': 'oci://dataflow-log@ociodscdev',
 'driver_shape': 'VM.Standard2.4',
 'executor_shape': 'VM.Standard2.4',
 'num_executors': 1}

```

```
run.oci_link
```

```
Saving processed data to jdbc:oracle:thin:@database_high?TNS_ADMIN=/tmp/
```

### Read from the Database Using PySpark

PySpark can be used to load data from an Oracle Autonomous Database (ADB) into a Spark application. The next cell makes a JDBC connection to the database defined using the `adb_url` variable, and accesses the table defined with `table_name`. The credentials stored in the vault and previously read into memory are used. After this command is run, you can perform Spark operations on it.

The table is relatively small so the notebook uses PySpark in the notebook session. However, for larger jobs, we recommended that you use the [Oracle](#) service.

```

if "adb_url" in globals():
    output_dataframe = sc.read \
        .format("jdbc") \
        .option("url", adb_url) \
        .option("dbtable", table_name) \
        .option("user", user) \
        .option("password", password) \

```

(continues on next page)

(continued from previous page)

```

        .load()
    else:
        print("Skipping as it appears that you do not have adb_url configured.")

```

The database table is loaded into Spark so that you can perform operations to transform, model, and more. In the next cell, the notebook prints the table demonstrating that it was successfully loaded into Spark from the ADB.

```

if "adb_url" in globals():
    output_dataframe.show()
else:
    print("Skipping as it appears that you do not have output_dataframe configured.")

```

```

+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Age| Attrition| TravelForWork| SalaryLevel| JobFunction| CommuteLength|
↪ EducationalLevel| EducationField| Directs| EmployeeNumber| EnvironmentSatisfaction|
↪ Gender| HourlyRate| JobInvolvement| JobLevel| JobRole| JobSatisfaction|
↪ MaritalStatus| MonthlyIncome| MonthlyRate| NumCompaniesWorked| Over18| OverTime|
↪ PercentSalaryHike| PerformanceRating| RelationshipSatisfaction| WeeklyWorkedHours|
↪ StockOptionLevel| YearsinIndustry| TrainingTimesLastYear| WorkLifeBalance| YearsOnJob|
↪ YearsAtCurrentLevel| YearsSinceLastPromotion| YearsWithCurrManager| name|
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪ +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 42| Yes| infrequent| 5054| Product Management| 2|
↪ L2| Life Sciences| 1| 1| 2| Female|
↪ 94| 3| 2| Sales Executive| 4| Single|
↪ 5993| 19479| 8| Y| Yes| 11|
↪ 3| 1| 80| 0|
↪ 8| 0| 1| 6| 4|
↪ 0| 5| Tracy Moore|
| 50| No| often| 1278| Software Developer| 9|
↪ L1| Life Sciences| 1| 2| 3| Male|
↪ 61| 2| 2| Research Scientist| 2| Married|
↪ 5130| 24907| 1| Y| No| 23|
↪ 4| 4| 80| 1|
↪ 10| 3| 3| 10| 7|
↪ 1| 7| Andrew Hoover|
| 38| Yes| infrequent| 6296| Software Developer| 3|
↪ L2| Other| 1| 4| 4| Male|
↪ 92| 2| 1| Laboratory Techni...| 3| Single|
↪ 2090| 2396| 6| Y| Yes| 15|
↪ 3| 2| 80| 0|

```

(continues on next page)

(continued from previous page)

→	7		3		3	0		0	→
→		0			0	Julie Bell			→
	34	No	often		6384	Software Developer	4		→
→	L4	Life Sciences	1		5		4	Female	→
→	56	3	1		Research Scientist		3	Married	→
→	2909	23159			1	Y	Yes	11	→
→	3				3		80	0	→
→	8		3		3		8	7	→
→		3			0	Thomas Adams			→
	28	No	infrequent		2710	Software Developer	3		→
→	L1	Medical	1		7		1	Male	→
→	40	3	1		Laboratory Techni...		2	Married	→
→	3468	16632			9	Y	No	12	→
→	3				4		80	1	→
→	6		3		3		2	2	→
→		2			2	Johnathan Burnett			→
	33	No	often		4608	Software Developer	3		→
→	L2	Life Sciences	1		8		4	Male	→
→	79	3	1		Laboratory Techni...		4	Single	→
→	3068	11864			0	Y	No	13	→
→	3				3		80	0	→
→	8		2		2		7	7	→
→		3			6	Rhonda Grant			→
	60	No	infrequent		6072	Software Developer	4		→
→	L3	Medical	1		10		3	Female	→
→	81	4	1		Laboratory Techni...		1	Married	→
→	2670	9964			4	Y	Yes	20	→
→	4				1		80	3	→
→	12		3		2		1	0	→
→		0			0	Brandon Gill			→
	31	No	infrequent		6228	Software Developer	25		→
→	L1	Life Sciences	1		11		4	Male	→
→	67	3	1		Laboratory Techni...		3	Divorced	→
→	2693	13335			1	Y	No	22	→
→	4				2		80	1	→
→	1		2		3		1	0	→
→		0			0	Debbie Chan			→
	39	No	often		990	Software Developer	24		→
→	L3	Life Sciences	1		12		4	Male	→
→	44	2	3		Manufacturing Dir...		3	Single	→
→	9526	8787			0	Y	No	21	→
→	4				2		80	0	→
→	10		2		3		9	7	→
→		1			8	Kayla Ward			→
	37	No	infrequent		5958	Software Developer	28		→
→	L3	Medical	1		13		3	Male	→
→	94	3	2		Healthcare Repres...		3	Married	→
→	5237	16577			6	Y	No	13	→
→	3				2		80	2	→
→	17		3		2		7	7	→
→		7			7	Angel Vaughn			→
	36	No	infrequent		3710	Software Developer	17		→

(continues on next page)

(continued from previous page)

→	L3	Medical	1	14	1	Male	→
→	84	4	1	Laboratory Techni...	2	Married	→
→	2426	16479		0  Y  No		13	→
→	3		3	80	1		→
→	6		5	3  5	4		→
→		0	3	Samantha Parker			→
	30	No  infrequent	700	Software Developer	16		→
→	L2	Life Sciences	1	15	4	Female	→
→	49	2	2	Laboratory Techni...	3	Single	→
→	4193	12682		0  Y  Yes		12	→
→	3		4	80	0		→
→	10		3	3  9	5		→
→		0	8	Melanie McBride			→
	32	No  infrequent	3072	Software Developer	27		→
→	L1	Life Sciences	1	16	1	Male	→
→	31	3	1	Research Scientist	3	Divorced	→
→	2911	15170		1  Y  No		17	→
→	3		4	80	1		→
→	5		1	2  5	2		→
→		4	3	Bradley Hall			→
	35	No  infrequent	6172	Software Developer	20		→
→	L2	Medical	1	18	2	Male	→
→	93	3	1	Laboratory Techni...	4	Divorced	→
→	2661	8758		0  Y  No		11	→
→	3		3	80	1		→
→	3		2	3  2	2		→
→		1	2	Patrick Lee			→
	29	Yes  infrequent	472	Software Developer	25		→
→	L3	Life Sciences	1	19	3	Male	→
→	50	2	1	Laboratory Techni...	3	Single	→
→	2028	12947		5  Y  Yes		14	→
→	3		2	80	0		→
→	6		4	3  4	2		→
→		0	3	Jessica Willis			→
	30	No  infrequent	6370	Software Developer	22		→
→	L4	Life Sciences	1	20	2	Female	→
→	51	4	3	Manufacturing Dir...	1	Divorced	→
→	9980	10195		1  Y  No		11	→
→	3		3	80	1		→
→	10		1	3  10	9		→
→		8	8	Chad Scott			→
	33	No  infrequent	1530	Software Developer	6		→
→	L2	Life Sciences	1	21	1	Male	→
→	80	4	1	Research Scientist	2	Divorced	→
→	3298	15053		0  Y  Yes		12	→
→	3		4	80	2		→
→	7		5	2  6	2		→
→		0	5	Gregory Bennett			→
	23	No  none	5150	Software Developer	17		→
→	L2	Medical	1	22	4	Male	→
→	96	4	1	Laboratory Techni...	4	Divorced	→
→	2935	7324		1  Y  Yes		13	→

(continues on next page)

(continued from previous page)

→		3		2		80		2	↵
→	1		2		2	1		0	↵
→		0		0	Jesse Palmer				↵
	54	No	infrequent	5590	Product Management		3		↵
→	L4	Life Sciences	1		23		1	Female	↵
→	78		2	4	Manager		4	Married	↵
→	15427	22021			2	Y	No	16	↵
→		3		3				0	↵
→	31		3		3	25		8	↵
→		3		7	Dr. Erin Good DDS				↵
	39	No	infrequent	1700	Software Developer		3		↵
→	L3	Life Sciences	1		24		4	Male	↵
→	45		3	1	Research Scientist		4	Single	↵
→	3944	4306			5	Y	Yes	11	↵
→		3		3				0	↵
→	6		3		3	3		2	↵
→		1		2	Kathy Patrick				↵
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----									
→	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----								
→	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----								
→	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----								
→	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----								
→	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----								
only showing top 20 rows									

### Cleaning Up Artifacts

This example created a number of artifacts, such as unzipping the wallet file, creating a database table, and starting a Spark cluster. Next, you remove these resources.

```

if wallet_path != "<wallet_path>":
    connection.update_repository(key="pyspark_adb", value=adb_creds)
    connection.import_wallet(wallet_path=wallet_path, key="pyspark_adb")
    conn = cx_Oracle.connect(user, password, tnsname)
    cursor = conn.cursor()
    cursor.execute(f"DROP TABLE {table_name}")
    cursor.close()
    conn.close()
else:
    print("Skipping as it appears that you do not have wallet_path specified.")

if "tns_path" in globals():
    shutil.rmtree(tns_path)

sc.stop()

```

### 13.7.6 Example Notebook: Using the ADB with PySpark

This notebook demonstrates how to use PySpark to process data in Object Storage, and save the results to an ADB. It also demonstrates how to query data from an ADB using a local PySpark session.

```
import base64
import cx_Oracle
import oci
import os
import shutil
import tempfile
import zipfile

from ads.database import connection
from ads.vault.vault import Vault
from pyspark import SparkConf
from pyspark.sql import SparkSession
from urllib.parse import urlparse
```

#### Introduction

It has become a common practice to store structured and semi-structured data using services such as Object Storage. This provides a scalable solution to store vast quantities of data that can be post-processed. However, using a relational database management system (RDMS) such as the Oracle ADB provides advantages like ACID compliance, rapid relational joins, support for complex business logic, and more. It is important to be able to access information stored in Object Storage, process that information, and load it into an RBMS. This notebook demonstrates how to use PySpark, a Python interface to Apache Spark, to perform these operations.

This notebook uses a publicly accessible Object Storage location to read from. However, an ADB needs to be configured with permissions to create a table, write to that table, and read from it. It also assumes that the credentials to access the database are stored in the Vault. This is the best practice as it prevents the credentials from being stored locally or in the notebook where they may be accessible to others. If you do not have credentials stored in the Vault. Once credentials to the database, are stored in the Vault, you need the OCIDs for the Vault, encryption key, and the secret.

ADB's have an additional level of security that is needed to access them and are wallet file. You can obtain the wallet file from your account administrator or download it using the steps that are outlined in the [downloading a wallet(<https://docs.oracle.com/en-us/iaas/Content/Database/Tasks/adbconnecting.htm#access>)]. The wallet file is a ZIP file. This notebook unzips the wallet and updates the configuration settings so you don't have to.

The database connection also needs the TNS name of the database. Your database administrator can give you the TNS name of the database that you have access to.

#### Setup the Required Variables

The required variables to set up are:

1. `vault_id`, `key_id`, `secret_ocid`: The OCID of the secret by storing the username and password required to connect to your ADB in a secret within the OCI Vault service. Note that the secret is the credential needed to access a database. This notebook is designed so that any secret can be stored as long as it is in the form of a dictionary. To store your secret, just modify the dictionary, see the `vault.ipynb` example notebook for detailed steps to generate this OCID.
2. `tnsname`: A TNS name valid for the database.
3. `wallet_path`: The local path to your wallet ZIP file, see the `autonomous_database.ipynb` example notebook for instructions on accessing the wallet file.

```
secret_ocid = "secret_ocid"
tnsname = "tnsname"
wallet_path = "wallet_path"
vault_id = "vault_id"
key_id = "key_id"
```

### Obtain Credentials from the Vault

If the `vault_id`, `key_id`, and `secret_id` have been updated, then the notebook obtains a handle to the vault with a variable called `vault`. This uses the `get_secret()` method to return a dictionary with the user credentials. The approach assumes that the Accelerated Data Science (ADS) library was used to store the secret.

```
if vault_id != "<vault_id>" and key_id != "<key_id>" and secret_ocid != "<secret_ocid>":
    print("Getting wallet username and password")
    vault = Vault(vault_id=vault_id, key_id=key_id)
    adb_creds = vault.get_secret(secret_ocid)
    user = adb_creds["username"]
    password = adb_creds["password"]
else:
    print("Skipping as it appears that you do not have vault, key, and secret ocid_
    ↪specified.")
```

Getting wallet username and password

### Setup the Wallet

An ADB requires a wallet file to access the database. The `wallet_path` variable defines the location of this file. The next cell prepares the wallet file to make a connection to the database. It also creates the ADB connection string, `adb_url`.

```
def setup_wallet(wallet_path):
    """
    Prepare ADB wallet file for use in PySpark.
    """

    temporary_directory = tempfile.mkdtemp()
    zip_file_path = os.path.join(temporary_directory, "wallet.zip")

    # Extract everything locally.
    with zipfile.ZipFile(wallet_path, "r") as zip_ref:
        zip_ref.extractall(temporary_directory)

    return temporary_directory

if wallet_path != "<wallet_path>":
    print("Setting up wallet")
    tns_path = setup_wallet(wallet_path)
else:
    print("Skipping as it appears that you do not have wallet_path specified.")
```

Setting up wallet



```
if "tns_path" in globals() and tnsname != "<tnsname>":
    adb_url = f"jdbc:oracle:thin:@{tnsname}?TNS_ADMIN={tns_path}"
else:
    print("Skipping, as the tns_path or tnsname are not defined.")
```

### Reading Data from Object Storage

This notebook uses PySpark to access the Object Storage file. The next cell creates a Spark application called “Python Spark SQL Example” and returns a SparkContext. The SparkContext, normally called `sc`, is a handle to the Spark application.

The data file that is used is relatively small so the notebook uses PySpark by running a version of Spark in local mode. That means, it is running in the notebook session. For larger jobs, we recommended that you use the [Oracle](#) service, which is an Oracle managed Spark service.

```
# create a spark session
sc = SparkSession \
    .builder \
    .appName("Python Spark SQL Example") \
    .getOrCreate()
```

This notebook reads in a data file that is stored in an Oracle Object Storage file. This is defined with the `file_path` variable. The SparkContext with the `read.option().csv()` methods is used to read in the CSV file from Object Storage into a data frame.

```
file_path = "oci://hosted-ds-datasets@bigdatadatasciencelarge/synthetic/orcl_attrition.
↳ csv"
input_dataframe = sc.read.option("header", "true").csv(file_path)
```

### Save the Data to the Database

This notebook creates a table in your database with the name specified with `table_name`. The name that is defined should be unique so that it does not interfere with any existing table in your database. If it does, change the value to something that is unique.

```
table_name = "ODSC_PYSPARK_ADB_DEMO"

if tnsname != "<tnsname>" and "adb_url" in globals():
    print("Saving processed data to " + adb_url)
    properties = {
        "oracle.net.tns_admin": tnsname,
        "password": password,
        "user": user,
    }
    input_dataframe.write.jdbc(
        url=adb_url, table=table_name, properties=properties
    )
else:
    print("Skipping as it appears that you do not have tnsname specified.")
```



## BIG DATA SERVICE

New in version 2.5.10.

The Oracle Big Data Service (BDS) is an Oracle Cloud Infrastructure (OCI) service designed for a diverse set of big data use cases and workloads. From short-lived clusters used to tackle specific tasks to long-lived clusters that manage data lakes. BDS scales to meet an organization's requirements at a low cost and with the highest levels of security. To be able to connect to the BDS from the notebook session, the cluster created must have Kerberos enabled.

### 14.1 Quick Start

#### 14.1.1 Set Up A Conda Environment

The following are the recommended steps to create a conda environment to connect to BDS:

- Open a terminal window then run the following commands:
- `odsc conda install -s pyspark30_p37_cpu_v5`: Install the PySpark conda environment.

#### 14.1.2 Connect from a Notebook

##### 14.1.2.1 Using the Vault

```
import ads
import os

from ads.bds.auth import krbcontext
from ads.secrets.big_data_service import BDSSecretKeeper
from pyhive import hive

ads.set_auth('resource_principal')
with BDSSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal=cred["principal"], keytab_path=cred['keytab_path']):
        cursor = hive.connect(host=cred["hive_host"],
                               port=cred["hive_port"],
                               auth='KERBEROS',
                               kerberos_service_name="hive").cursor()
```

### 14.1.2.2 Without Using the Vault

```
import ads
import fsspec
import os

from ads.bds.auth import refresh_ticket

ads.set_auth('resource_principal')
refresh_ticket(principal="<your_principal>", keytab_path="<your_local_keytab_file_path>",
              kerb5_path="<your_local_kerb5_config_file_path>")
cursor = hive.connect(host="<hive_host>", port="<hive_port>",
                     auth='KERBEROS', kerberos_service_name="hive").cursor()
```

## 14.2 Conda Environment

To work with BDS in a notebook session or job, you must have a conda environment that supports the BDS module in ADS along with support for PySpark. This section demonstrates how to modify a PySpark Data Science conda environment to work with BDS. It also demonstrates how to publish this conda environment so that you can share it with team members and use it in jobs.

### 14.2.1 Create

The following are the recommended steps to create a conda environment to connect to BDS:

- Open a terminal window then run the following commands:
- `odsc conda install -s pyspark30_p37_cpu_v5`: Install the PySpark conda environment.

### 14.2.2 Publish

- Create an Object Storage bucket to store published conda environments.
- Open a terminal window then run the following commands and actions:
- `odsc conda init -b <bucket_name> -b <namespace> -a <resource_principal or api_key>`: Initialize the environment so that you can work with Published Conda Environments.
- `odsc conda publish -s pyspark30_p37_cpu_v3`: Publish the conda environment.
- In the OCI Console, open Data Science.
- Select a project.
- Select a click the notebook session's name, or the Actions menu, and click Open to open the notebook session's JupyterLab interface in another tab..
- Click Published Conda Environments in the Environment Explorer tab to list all the published conda environments that are available in your designated Object Storage bucket.
- Select the Environment Version that you specified.
- Click the copy button adjacent to the Source conda environment to copy the file source path to use when installing the conda environment in other notebook sessions or to use with jobs.

## 14.3 Connect

### 14.3.1 Notebook Session

Notebook sessions require a conda environment that has the BDS module of ADS installed.

#### 14.3.1.1 Using the Vault

The preferred method to connect to a BDS cluster is to use the `BDSecretKeeper` class. This allows you to store the BDS credentials in the vault and not the notebook. It also provides a greater level of access control to the secrets and allows for credential rotation without breaking connections from various sources.

```
import ads
import os

from ads.bds.auth import krbcontext
from ads.secrets.big_data_service import BDSecretKeeper
from pyhive import hive

ads.set_auth('resource_principal')
with BDSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal=cred["principal"], keytab_path=cred['keytab_path']):
        cursor = hive.connect(host=cred["hive_host"],
                               port=cred["hive_port"],
                               auth='KERBEROS',
                               kerberos_service_name="hive").cursor()
```

#### 14.3.1.2 Without Using the Vault

BDS requires a Kerberos ticket to authenticate to the service. The preferred method is to use the vault and `BDSecretKeeper` because it is more secure, and prevents private information from being stored in a notebook. However, if this is not possible, you can use the `refresh_ticket()` method to manually create the Kerberos ticket. This method requires the following parameters:

- `kerb5_path`: The path to the `krb5.conf` file. You can copy this file from the master node of the BDS cluster located in `/etc/krb5.conf`.
- `keytab_path`: The path to the principal's keytab file. You can download this file from the master node on the BDS cluster.
- `principal`: The unique identity to that Kerberos can assign tickets to.

```
import ads
import fsspec
import os

from ads.bds.auth import refresh_ticket

ads.set_auth('resource_principal')
refresh_ticket(principal="<your_principal>", keytab_path="<your_local_keytab_file_path>",
               kerb5_path="<your_local_kerb5_config_file_path>")
cursor = hive.connect(host="<hive_host>", port="<hive_port>",
                      auth='KERBEROS', kerberos_service_name="hive").cursor()
```

### 14.3.2 Jobs

A job requires a conda environment that has the BDS module of ADS installed. It also requires secrets and configuration information that can be used to obtain a Kerberos ticket for authentication. You must copy the `keytab` and `krb5.conf` files to the jobs instance and can be copied as part of the job. We recommend that you save them into the vault then use `BDSecretKeeper` to access them. This is secure because the vault provides access control and allows for key rotation without breaking existing jobs. You can use the notebook to load configuration parameters like `hdfs_host`, `hdfs_port`, `hive_host`, `hive_port`, and so on. The `keytab` and `krb5.conf` files are securely loaded from the vault then saved in the jobs instance. The `krbcontext()` method is then used to create the Kerberos ticket. Once the ticket is created, you can query BDS.

## 14.4 File Management

This section demonstrates various methods to work with files on BDS' HDFS, see the individual framework's documentation for details.

A Kerberos ticket is needed to *connect to the BDS cluster*. This authentication ticket can be obtained with the `refresh_ticket()` method or with the use of the Vault and a `BDSecretKeeper` object. This section will demonstrate the use of the `BDSecretKeeper` object as this is more secure and is the preferred method.

### 14.4.1 FSSpec

The `fsspec` or `Filesystem Spec` is an interface that allows access to local, remote, and embedded file systems. You use it to access data stored in the BDS' HDFS. This connection is made with the `WebHDFS` protocol.

The `fsspec` library must be able to access BDS so a Kerberos ticket must be generated. The secure and recommended method to do this is to use `BDSecretKeeper` that stores the BDS credentials in the vault not the notebook session.

This section outlines some common file operations, see the `fsspec` [API Reference](#) for complete details on the features that are demonstrated and additional functionality.

*Pandas* and *PyArrow* can also use `fsspec` to perform file operations.

#### 14.4.1.1 Connect

Credentials and configuration information is stored in the vault. This information is used to obtain a Kerberos ticket and define the `hdfs_config` dictionary. This configuration dictionary is passed to the `fsspec.filesystem()` method to make a connection to the BDS' underlying HDFS storage.

```
import ads
import fsspec

from ads.secrets.big_data_service import BDSecretKeeper
from ads.bds.auth import has_kerberos_ticket, krbcontext

ads.set_auth("resource_principal")
with BDSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal = cred["principal"], keytab_path = cred['keytab_path']):
        hdfs_config = {
            "protocol": "webhdfs",
            "host": cred["hdfs_host"],
            "port": cred["hdfs_port"],
```

(continues on next page)

(continued from previous page)

```

        "kerberos": "True"
    }

fs = fsspec.filesystem(**hdfs_config)

```

#### 14.4.1.2 Delete

Delete files from HDFS using the `.rm()` method. It accepts a path of the files to delete.

```
fs.rm("/data/biketrips/2020??-tripdata.csv", recursive=True)
```

#### 14.4.1.3 Download

Download files from HDFS to a local storage device using the `.get()` method. It takes the HDFS path of the files to download, and the local path to store the files.

```
fs.get("/data/biketrips/20190[123456]-tripdata.csv", local_path="./first_half/",
      overwrite=True)
```

#### 14.4.1.4 List

The `.ls()` method lists files. It returns the matching file names as a list.

```
fs.ls("/data/biketrips/2019??-tripdata.csv")
```

```

['201901-tripdata.csv',
 '201902-tripdata.csv',
 '201903-tripdata.csv',
 '201904-tripdata.csv',
 '201905-tripdata.csv',
 '201906-tripdata.csv',
 '201907-tripdata.csv',
 '201908-tripdata.csv',
 '201909-tripdata.csv',
 '201910-tripdata.csv',
 '201911-tripdata.csv',
 '201912-tripdata.csv']

```

#### 14.4.1.5 Upload

The `.put()` method is used to upload files from local storage to HDFS. The first parameter is the local path of the files to upload. The second parameter is the HDFS path where the files are to be stored. `.upload()` is an alias of `.put()`. .. code-block:: python3

```

fs.put(
    lpath="./first_half/20200[456]-tripdata.csv", rpath="/data/biketrips/second_quarter/"
)

```

### 14.4.2 Ibis

**Ibis** is an open-source library by [Cloudera](#) that provides a Python framework to access data and perform analytical computations from different sources. Ibis allows access to the data using HDFS. You use the `ibis.impala.hdfs_connect()` method to make a connection to HDFS, and it returns a handler. This handler has methods such as `.ls()` to list, `.get()` to download, `.put()` to upload, and `.rm()` to delete files. These operations support globbing. Ibis' HDFS connector supports a variety of [additional operations](#).

#### 14.4.2.1 Connect

After obtaining a Kerberos ticket, the `hdfs_connect()` method allows access to the HDFS. It is a thin wrapper around a `fspec` file system. Depending on your system configuration, you may need to define the `ibis.options.impala.temp_db` and `ibis.options.impala.temp_hdfs_path` options.

```
import ibis

with BDSSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal=cred["principal"], keytab_path=cred['keytab_path']):
        hdfs = ibis.impala.hdfs_connect(host=cred['hdfs_host'], port=cred['hdfs_port'],
                                         use_https=False, verify=False,
                                         auth_mechanism='GSSAPI', protocol='webhdfs')
```

#### 14.4.2.2 Delete

Delete files from HDFS using the `.rm()` method. It accepts a path of the files to delete.

```
hdfs.rm("/data/biketrips/2020??-tripdata.csv", recursive=True)
```

#### 14.4.2.3 Download

Download files from HDFS to a local storage device using the `.get()` method. It takes the HDFS path of the files to download, and the local path to store the files.

```
hdfs.get("/data/biketrips/20190[123456]-tripdata.csv", local_path="./first_half/",
↪ overwrite=True)
```

#### 14.4.2.4 List

The `.ls()` method lists files. It returns the matching file names as a list.

```
hdfs.ls("/data/biketrips/2019??-tripdata.csv")
```

```
['201901-tripdata.csv',
 '201902-tripdata.csv',
 '201903-tripdata.csv',
 '201904-tripdata.csv',
 '201905-tripdata.csv',
 '201906-tripdata.csv',
 '201907-tripdata.csv',
```

(continues on next page)



(continued from previous page)

```
'201908-tripdata.csv',
'201909-tripdata.csv',
'201910-tripdata.csv',
'201911-tripdata.csv',
'201912-tripdata.csv']
```

#### 14.4.2.5 Upload

Use the `.put()` method to upload files from local storage to HDFS. The first parameter is the HDFS path where the files are to be stored. The second parameter is the local path of the files to upload.

```
hdfs.put(rpath="/data/biketrips/second_quarter/",
        lpath="./first_half/20200[456]-tripdata.csv",
        overwrite=True, recursive=True)
```

### 14.4.3 Pandas

Pandas allows access to BDS' HDFS system through :ref: *FSSpec*. This section demonstrates some common operations.

#### 14.4.3.1 Connect

```
import ads
import fsspec

from ads.secrets.big_data_service import BDSSecretKeeper
from ads.bds.auth import has_kerberos_ticket, krbcontext

ads.set_auth("resource_principal")
with BDSSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal = cred["principal"], keytab_path = cred['keytab_path']):
        hdfs_config = {
            "protocol": "webhdfs",
            "host": cred["hdfs_host"],
            "port": cred["hdfs_port"],
            "kerberos": "True"
        }

fs = fsspec.filesystem(**hdfs_config)
```

### 14.4.3.2 File Handle

You can use the `fsspec.open()` method to open a data file. It returns a file handle. That file handle, `f`, can be passed to any Pandas' methods that support file handles. In this example, a file on a BDS' HDFS cluster is read into a Pandas dataframe.

```
with fs.open("/data/biketrips/201901-tripdata.csv", "r") as f:
    df = pd.read_csv(f)
```

### 14.4.3.3 URL

Pandas supports `fsspec` so you can preform file operations by specifying a protocol string. The WebHDFS protocol is used to access files on BDS' HDFS system. The protocol string has this format:

```
webhdfs://host:port/path/to/data
```

The host and port parameters can be passed in the protocol string as follows:

```
df = pd.read_csv(f"webhdfs://{hdfs_config['host']}:{hdfs_config['port']}/data/biketrips/
↳201901-tripdata.csv",
                 storage_options={'kerberos': 'True'})
```

You can also pass the host and port parameters in the dictionary used by the `storage_options` parameter. The sample code for `hdfs_config` defines the host and port with the keys `host` and `port` respectively.

```
hdfs_config = {
    "protocol": "webhdfs",
    "host": cred["hdfs_host"],
    "port": cred["hdfs_port"],
    "kerberos": "True"
}
```

In this case, Pandas uses the following syntax to read a file on BDS' HDFS cluster:

```
df = pd.read_csv(f"webhdfs:///data/biketrips/201901-tripdata.csv",
                 storage_options=hdfs_config)
```

## 14.4.4 PyArrow

[PyArrow](#) is a Python interface to [Apache Arrow](#). Apache Arrow is an in-memory columnar analytical tool that is designed to process data at scale. PyArrow supports the `fspec.filesystem()` through the use of the `filesystem` parameter in many of its data operation methods.

#### 14.4.4.1 Connect

Make a connection to BDS' HDFS using fsspec:

```
import ads
import fsspec

from ads.secrets.big_data_service import BDSSecretKeeper
from ads.bds.auth import has_kerberos_ticket, krbcontext

ads.set_auth("resource_principal")
with BDSSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal = cred["principal"], keytab_path = cred['keytab_path']):
        hdfs_config = {
            "protocol": "webhdfs",
            "host": cred["hdfs_host"],
            "port": cred["hdfs_port"],
            "kerberos": "True"
        }

fs = fsspec.filesystem(**hdfs_config)
```

#### 14.4.4.2 Filesystem

The following sample code shows several different PyArrow methods for working with BDS' HDFS using the filesystem parameter:

```
import pyarrow as pa
import pyarrow.parquet as pq
import pyarrow.dataset as ds

ds = ds.dataset("/path/on/BDS/HDFS/data.csv", format="csv", filesystem=fs)
pq.write_table(ds.to_table(), '/path/on/BDS/HDFS/data.parquet', filesystem=fs)

import pandas as pd
import numpy as np

idx = pd.date_range('2022-01-01 12:00:00.000', '2022-03-01 12:00:00.000', freq='T')

df = pd.DataFrame({
    'numeric_col': np.random.rand(len(idx)),
    'string_col': pd._testing.rands_array(8, len(idx))},
    index = idx
)
df["dt"] = df.index
df["dt"] = df["dt"].dt.date

table = pa.Table.from_pandas(df)
pq.write_to_dataset(table, root_path="/path/on/BDS/HDFS", partition_cols=["dt"],
                    flavor="spark", filesystem=fs)
```

## 14.5 SQL Data Management

This section demonstrates how to perform standard SQL-based data management operations in BDS using various frameworks, see the individual framework's documentation for details.

A Kerberos ticket is needed to *connect to the BDS cluster*. You can obtain this authentication ticket with the `refresh_ticket()` method, or with the use of the vault and a `BDSecretKeeper` object. This section demonstrates the use of the `BDSecretKeeper` object because this is more secure and is the recommended method.

### 14.5.1 Ibis

`Ibis` is an open-source library by [Cloudera](#) that provides a Python framework to access data and perform analytical computations from different sources. The `Ibis` project is designed to provide an abstraction over different dialects of SQL. It enables the data scientist to interact with many different data systems. Some of these systems are Dask, MySQL, Pandas, PostgreSQL, PySpark, and most importantly for use with BDS, Hadoop clusters.

#### 14.5.1.1 Connect

Obtaining a Kerberos ticket, depending on your system configuration, you may need to define the `ibis.options.impala.temp_db` and `ibis.options.impala.temp_hdfs_path` options. The `ibis.impala.connect()` method makes a connection to the [Impala execution backend](#). The `.sql()` allows you to run SQL commands on the data.

```
import ibis

with BDSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal=cred["principal"], keytab_path=cred['keytab_path']):
        ibis.options.impala.temp_db = '<temp_db>'
        ibis.options.impala.temp_hdfs_path = '<temp_hdfs_path>'
        hdfs = ibis.impala.hdfs_connect(host=cred['hdfs_host'], port=cred['hdfs_port'],
                                         use_https=False, verify=False,
                                         auth_mechanism='GSSAPI', protocol='webhdfs')
        client = ibis.impala.connect(host=cred['hive_host'], port=cred['hive_port'],
                                     hdfs_client=hdfs, auth_mechanism="GSSAPI",
                                     use_ssl=False, kerberos_service_name="hive")
```

#### 14.5.1.2 Query

To query the data using `ibis` use an SQL DML command like `SELECT`. Pass the string to the `.sql()` method, and then call `.execute()` on the returned object. The output is a Pandas dataframe.

```
df = client.sql("SELECT * FROM bikes.trips LIMIT 100").execute(limit=None)
```

### 14.5.1.3 Close a Connection

It is important to close sessions when you don't need them anymore. This frees up resources in the system. Use the `.close()` method close sessions.

```
client.close()
```

## 14.5.2 Impala

[Impala](#) is a Python client for [HiveServer2](#) implementations (i.e. Impala, Hive). Both Impala and PyHive clients are HiveServer2 compliant so the connection syntax is very similar. The difference is that the Impala client uses the Impala query engine and PyHive uses Hive. In practical terms, Hive is best suited for long-running batch queries and Impala is better suited for real-time interactive querying, see [more about the differences between Hive and Impala](#).

The Impala `dbapi` module is a [Python DB-API](#) interface.

### 14.5.2.1 Connect

After obtaining a Kerberos ticket, use the `connect()` method to make the connection. It returns a connection, and the `.cursor()` method returns a cursor object. The cursor has the method `.execute()` that allows you to run Impala SQL commands on the data.

```
from impala.dbapi import connect

with BDSSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal=cred["principal"], keytab_path=cred['keytab_path']):
        cursor = connect(host=cred["hive_host"], port=cred["hive_port"],
                        auth_mechanism="GSSAPI", kerberos_service_name="hive").cursor()
```

### 14.5.2.2 Create a Table

To create an Impala table and insert data, use the `.execute()` method on the cursor object, and pass in Impala SQL commands to perform these operations.

```
cursor.execute("CREATE TABLE default.location (city STRING, province STRING)")
cursor.execute("INSERT INTO default.location VALUES ('Halifax', 'Nova Scotia')")
```

### 14.5.2.3 Query

To query an Impala table, use an Impala SQL DML command like `SELECT`. Pass this string to the `.execute()` method on the cursor object to create a record set in the cursor. You can obtain a Pandas dataframe with the `as_pandas()` function.

```
from impala.util import as_pandas

cursor.execute("SELECT * FROM default.location")
df = as_pandas(cursor)
```

#### 14.5.2.4 Drop a Table

To drop an Impala table, use an Impala SQL DDL command like `DROP TABLE`. Pass this string to the `.execute()` method on the cursor object.

```
cursor.execute("DROP TABLE IF EXISTS default.location")
```

#### 14.5.2.5 Close a Connection

It is important to close sessions when you don't need them anymore. This frees up resources in the system. Use the `.close()` method on the cursor object to close a connection.

```
cursor.close()
```

### 14.5.3 PyHive

PyHive is a set of interfaces to Presto and Hive. It is based on the [SQLAlchemy](#) and [Python DB-API](#) interfaces for Presto and Hive.

#### 14.5.3.1 Connect

After obtaining a Kerberos ticket, call the `hive.connect()` method to make the connection. It returns a connection, and the `.cursor()` method returns a cursor object. The cursor has the `.execute()` method that allows you to run Hive SQL commands on the data.

```
import ads
import os

from ads.bds.auth import krbcontext
from ads.secrets.big_data_service import BDSSecretKeeper
from pyhive import hive

ads.set_auth('resource_principal')
with BDSSecretKeeper.load_secret("<secret_id>") as cred:
    with krbcontext(principal=cred["principal"], keytab_path=cred['keytab_path']):
        cursor = hive.connect(host=cred["hive_host"],
                               port=cred["hive_port"],
                               auth='KERBEROS',
                               kerberos_service_name="hive").cursor()
```

#### 14.5.3.2 Create a Table

To create a Hive table and insert data, use the `.execute()` method on the cursor object and pass in Hive SQL commands to perform these operations.

```
cursor.execute("CREATE TABLE default.location (city STRING, province STRING)")
cursor.execute("INSERT INTO default.location VALUES ('Halifax', 'Nova Scotia')")
```

### 14.5.3.3 Query

To query a Hive table, use a Hive SQL DML command like `SELECT`. Pass this string to the `.execute()` method on the cursor object. This creates a record set in the cursor. You can access the actual records with methods like `.fetchall()`, `.fetchmany()`, and `.fetchone()`.

In the following example, the `.fetchall()` method is used in a `pd.DataFrame()` call to return all the records in Pandas dataframe: .

```
import pandas as pd

cursor.execute("SELECT * FROM default.location")
df = pd.DataFrame(cursor.fetchall(), columns=[col[0] for col in cursor.description])
```

### 14.5.3.4 Drop a Table

To drop a Hive table, use a Hive SQL DDL command like `DROP TABLE`. Pass this string to the `.execute()` method on the cursor object.

```
cursor.execute("DROP TABLE IF EXISTS default.location")
```

### 14.5.3.5 Close a Connection

It is important to close sessions when you don't need them anymore. This frees up resources in the system. Use the `.close()` method on the cursor object to close a connection.

```
cursor.close()
```





## DATA SCIENCE JOBS

Oracle Cloud Infrastructure (OCI) Data Science jobs enable you to define and run a repeatable machine learning task on a fully managed infrastructure, such as **data preparation, model training, hyperparameter optimization, batch inference, and so on**.

### 15.1 Overview

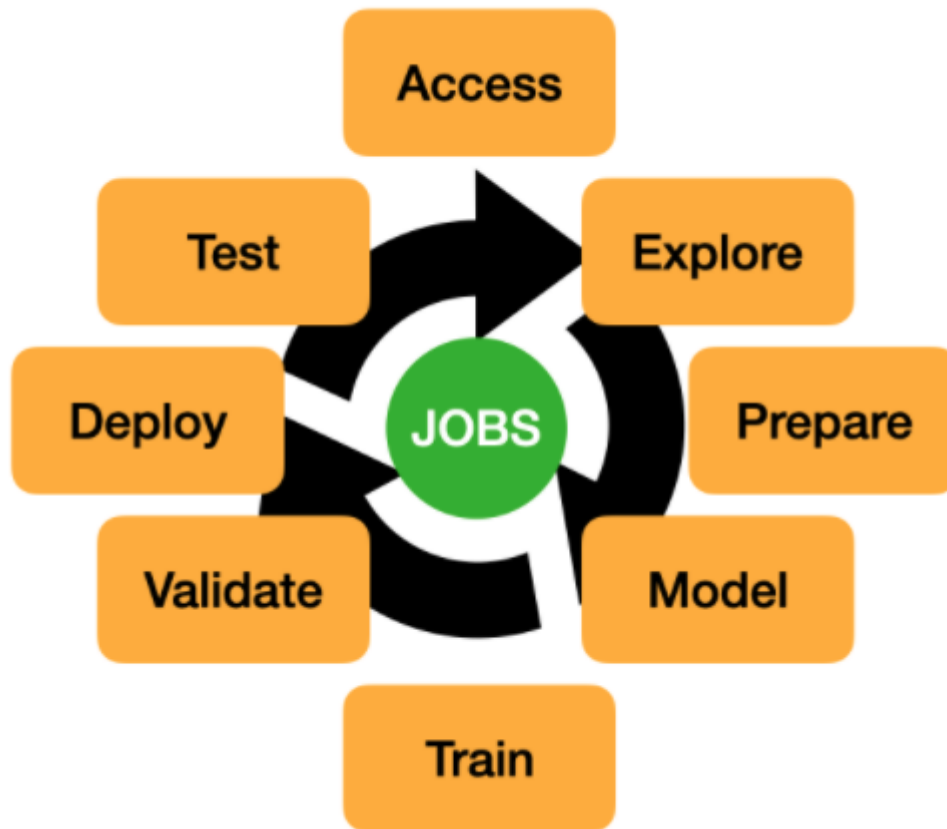
Data Science jobs allow you to run customized tasks outside of a notebook session. You can have Compute on demand and only pay for the Compute that you need. With jobs, you can run applications that perform tasks such as data preparation, model training, hyperparameter tuning, and batch inference. When the task is complete the compute automatically terminates. You can use the Logging service to capture output messages.

Using jobs, you can:

- Run machine learning (ML) or data science tasks outside of your JupyterLab notebook session.
- Operationalize discrete data science and machine learning tasks, such as reusable runnable operations.
- Automate your MLOps or CI/CD pipeline.
- Run batch or workloads triggered by events or actions.
- Batch, mini batch, or distributed batch job inference.
- In a JupyterLab notebook session, you can launch long running tasks or computation intensive tasks in a Data Science job to keep your notebook free for you to continue your work.

Typically, an ML and data science project is a series of steps including:

- Access
- Explore
- Prepare
- Model
- Train
- Validate
- Deploy
- Test



After the steps are completed, you can automate the process of data exploration, model training, deploying, and testing using jobs. A single change in the data preparation or model training to experiment with hyperparameter tunings can be run as a job and independently tested.

Data Science jobs consist of two types of resources: job and job run.

### 15.1.1 Job

A job is a template that describes the task. It contains elements like the job artifact, which is immutable. It can't be modified after being registered as a Data Science job. A job contains information about the Compute shape, logging configuration, Block Storage, and other options. You can configure environment variables that are used at run-time by the job run. You can also pass in CLI arguments. This allows a job run to be customized while using the same job as a template. You can override the environment variable and CLI parameters in job runs. Only the job artifact is immutable though the settings can be changed.

### 15.1.2 Job Run

A job run is an instantiation of a job. In each job run, you can override some of the job configuration. The most common configurations to change are the environment variables and CLI arguments. You can use the same job as a template and launch multiple simultaneous job runs to parallelize a large task. You can also sequence jobs and keep the state by writing state information to Object Storage.

For example, you could experiment with how different model classes perform on the same training data by using the ADSTuner to perform hyperparameter tuning on each model class. You could do this in parallel by having a different job run for each class of models. For a given job run, you could pass an environment variable that identifies the model class that you want to use. Each model can write its results to the Logging service or Object Storage. Then you can run a final sequential job that uses the best model class, and trains the final model on the entire dataset.

### 15.1.3 ADS Jobs

ADS jobs API calls separate the job configurations into infrastructure and runtime. Infrastructure specifies the configurations of the OCI resources and service for running the job. Runtime specifies the source code and the software environments for running the job. These two types of infrastructure are supported: [Data Science job](#) and [Data Flow](#).

## 15.2 Data Science Job

This section shows how you can use the ADS jobs APIs to run OCI Data Science jobs. You can use similar APIs to [Run a OCI DataFlow Application](#).

Before creating a job, ensure that you have policies configured for Data Science resources, see [About Data Science Policies](#).

### 15.2.1 Infrastructure

The Data Science job infrastructure is defined by a `DataScienceJob` instance. When creating a job, you specify the compartment ID, project ID, subnet ID, Compute shape, Block Storage size, log group ID, and log ID in the `DataScienceJob` instance. For example:

```
from ads.jobs import DataScienceJob

infrastructure = (
    DataScienceJob()
    .with_compartment_id("<compartment_ocid>")
    .with_project_id("<project_ocid>")
    .with_subnet_id("<subnet_ocid>")
    .with_shape_name("VM.Standard.E3.Flex")
    .with_shape_config_details(memory_in_gbs=16, ocpus=1) # Applicable only for the
↪ flexible shapes
    .with_block_storage_size(50)
    .with_log_group_id("<log_group_ocid>")
    .with_log_id("<log_ocid>")
)
```

If you are using these API calls in a Data Science [Notebook Session](#), and you want to use the same infrastructure configurations as the notebook session, you can initialize the `DataScienceJob` with only the logging configurations:

```
from ads.jobs import DataScienceJob

infrastructure = (
    DataScienceJob()
    .with_log_group_id("<log_group_ocid>")
    .with_log_id("<log_ocid>")
)
```

In some cases, you may want to override the shape and block storage size. For example, if you are testing your code in a CPU notebook session, but want to run the job in a GPU VM:

```
from ads.jobs import DataScienceJob

infrastructure = (
    DataScienceJob()
    .with_shape_name("VM.GPU2.1")
    .with_log_group_id("<log_group_ocid>")
    .with_log_id("<log_ocid>")
)
```

Data Science jobs support the following shapes:

Shape Name	Core Count	Memory (GB)
VM.Optimized3.Flex	18	256
VM.Standard3.Flex	32	512
VM.Standard.E4.Flex	16	1024
VM.Standard2.1	1	15
VM.Standard2.2	2	30
VM.Standard2.4	4	60
VM.Standard2.8	8	120
VM.Standard2.16	16	240
VM.Standard2.24	24	320
VM.GPU2.1	12	72
VM.GPU3.1	6	90
VM.GPU3.2	12	180
VM.GPU3.4	24	360

You can get a list of currently supported shapes by calling `DataScienceJob.instance_shapes()`.

## 15.2.2 Logs

In the preceding examples, both the log OCID and corresponding log group OCID are specified in the `DataScienceJob` instance. If your administrator configured the permission for you to search for logging resources, you can skip specifying the log group OCID because ADS automatically retrieves it.

If you specify only the log group OCID and no log OCID, a new Log resource is automatically created within the log group to store the logs, see [ADS Logging](#).

### 15.2.3 Runtime

A job can have different types of *runtime* depending on the source code you want to run:

- `ScriptRuntime` allows you to run Python, Bash, and Java scripts from a single source file (`.zip` or `.tar.gz`) or code directory, see [Run a Script](#) and [Run a ZIP file or folder](#).
- `PythonRuntime` allows you to run Python code with additional options, including setting a working directory, adding python paths, and copying output files, see [Run a ZIP file or folder](#).
- `NotebookRuntime` allows you to run a JupyterLab Python notebook, see [Run a Notebook](#).
- `GitPythonRuntime` allows you to run source code from a Git repository, see [Run from Git](#).

All of these runtime options allow you to configure a [Data Science Conda Environment](#) for running your code. For example, to define a python script as a job runtime with a TensorFlow conda environment you could use:

```
from ads.jobs import ScriptRuntime

runtime = (
    ScriptRuntime()
    .with_source("oci://bucket_name@namespace/path/to/script.py")
    .with_service_conda("tensorflow26_p37_cpu_v2")
)
```

You can store your source code in a local file path or location supported by `fspec`, including OCI Object Storage.

You can also use a custom conda environment published to OCI Object Storage by passing the `uri` to the `with_custom_conda()` method, for example:

```
runtime = (
    ScriptRuntime()
    .with_source("oci://bucket_name@namespace/path/to/script.py")
    .with_custom_conda("oci://bucket@namespace/conda_pack/pack_name")
)
```

For more details on custom conda environment, see [Publishing a Conda Environment to an Object Storage Bucket in Your Tenancy](#).

You can also configure the environment variables, command line arguments, and free form tags for runtime:

```
runtime = (
    ScriptRuntime()
    .with_source("oci://bucket_name@namespace/path/to/script.py")
    .with_service_conda("tensorflow26_p37_cpu_v2")
    .with_environment_variable(ENV="value")
    .with_argument("argument", key="value")
    .with_freeform_tag(tag_name="tag_value")
)
```

With the preceding arguments, the script is started as `python script.py argument --key value`.

## 15.2.4 Define a Job

With runtime and infrastructure, you can define a job and give it a name:

```
from ads.jobs import Job

job = (
    Job(name="<job_display_name>")
    .with_infrastructure(infrastructure)
    .with_runtime(runtime)
)
```

If the job name is not specified, a name is generated automatically based on the name of the job artifact and a time stamp.

Alternatively, a job can also be defined with keyword arguments:

```
job = Job(
    name="<job_display_name>",
    infrastructure=infrastructure,
    runtime=runtime
)
```

## 15.2.5 Create and Run

You can call the `create()` method of a job instance to create a job. After the job is created, you can call the `run()` method to create and start a job run. The `run()` method returns a `DataScienceJobRun`. You can monitor the job run output by calling the `watch()` method of the `DataScienceJobRun` instance:

```
# Create a job
job.create()
# Run a job, a job run will be created and started
job_run = job.run()
# Stream the job run outputs
job_run.watch()
```

```
2021-10-28 17:17:58 - Job Run ACCEPTED
2021-10-28 17:18:07 - Job Run ACCEPTED, Infrastructure provisioning.
2021-10-28 17:19:19 - Job Run ACCEPTED, Infrastructure provisioned.
2021-10-28 17:20:48 - Job Run ACCEPTED, Job run bootstrap starting.
2021-10-28 17:23:41 - Job Run ACCEPTED, Job run bootstrap complete. Artifact execution.
↪starting.
2021-10-28 17:23:50 - Job Run IN_PROGRESS, Job run artifact execution in progress.
2021-10-28 17:23:50 - <Log Message>
2021-10-28 17:23:50 - <Log Message>
2021-10-28 17:23:50 - ...
```

### 15.2.6 Override Configuration

When you run `job.run()`, the job is run with the default configuration. You may want to override this default configuration with custom variables. You can specify a custom job run display name, override command line argument, add additional environment variables, or free form tags as in this example:

```
job_run = job.run(
    name="<my_job_run_name>",
    args="new_arg --new_key new_val",
    env_var={"new_env": "new_val"},
    freeform_tags={"new_tag": "new_tag_val"}
)
```

### 15.2.7 YAML Serialization

A job instance can be serialized to a YAML file by calling `to_yaml()`, which returns the YAML as a string. You can easily share the YAML with others, and reload the configurations by calling `from_yaml()`. The `to_yaml()` and `from_yaml()` methods also take an optional `uri` argument for saving and loading the YAML file. This argument can be any URI to the file location supported by `fsspec`, including Object Storage. For example:

```
# Save the job configurations to YAML file
job.to_yaml(uri="oci://bucket_name@namespace/path/to/job.yaml")

# Load the job configurations from YAML file
job = Job.from_yaml(uri="oci://bucket_name@namespace/path/to/job.yaml")

# Save the job configurations to YAML in a string
yaml_string = job.to_yaml()

# Load the job configurations from a YAML string
job = Job.from_yaml("""
kind: job
spec:
  infrastructure:
    kind: infrastructure
    ...
""")
```

Here is an example of a YAML file representing the job defined in the preceding examples:

```
kind: job
spec:
  name: <job_display_name>
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    spec:
      logGroupId: <log_group_ocid>
      logId: <log_ocid>
      compartmentId: <compartment_ocid>
      projectId: <project_ocid>
      subnetId: <subnet_ocid>
```

(continues on next page)

(continued from previous page)

```

    shapeName: VM.Standard.E3.Flex
    shapeConfigDetails:
      memoryInGBs: 16
      ocpus: 1
      blockSizeSize: 50
  runtime:
    kind: runtime
    type: script
    spec:
      conda:
        slug: tensorflow26_p37_cpu_v2
        type: service
      scriptPathURI: oci://bucket_name@namespace/path/to/script.py

```

**ADS Job YAML schema**

```

kind:
  required: true
  type: string
  allowed:
    - job
spec:
  required: true
  type: dict
  schema:
    id:
      required: false
    infrastructure:
      required: false
    runtime:
      required: false
    name:
      required: false
      type: string

```

**Data Science Job Infrastructure YAML Schema**

```

kind:
  required: true
  type: "string"
  allowed:
    - "infrastructure"
type:
  required: true
  type: "string"
  allowed:
    - "dataScienceJob"
spec:
  required: true
  type: "dict"
  schema:
    blockSizeSize:

```

(continues on next page)



(continued from previous page)

```

    default: 50
    min: 50
    required: false
    type: "integer"
  compartmentId:
    required: false
    type: "string"
  displayName:
    required: false
    type: "string"
  id:
    required: false
    type: "string"
  logGroupId:
    required: false
    type: "string"
  logId:
    required: false
    type: "string"
  projectId:
    required: false
    type: "string"
  shapeName:
    required: false
    type: "string"
  subnetId:
    required: false
    type: "string"
  shapeConfigDetails:
    required: false
    type: "dict"

```

## 15.3 Run a Container

The ADS ContainerRuntime class allows you to run a container image using OCI data science jobs.

To use the ContainerRuntime, you need to first push the image to [OCI container registry](#). See [Creating a Repository](#) and [Pushing Images Using the Docker CLI](#) for more details.

### 15.3.1 Python

To configure ContainerRuntime, you must specify the container image. Similar to other runtime, you can add environment variables. You can optionally specify the *entrypoint* and *cmd* for running the container (See [Understand how CMD and ENTRYPOINT interact](#)).

```

from ads.jobs import Job, DataScienceJob, ContainerRuntime

job = (
    Job()
    .with_infrastructure(

```

(continues on next page)

(continued from previous page)

```

DataScienceJob()
  .with_log_group_id("<log_group_ocid>")
  .with_log_id("<log_ocid>")
  # The following infrastructure configurations are optional
  # if you are in an OCI data science notebook session.
  # The configurations of the notebook session will be used as defaults
  .with_compartment_id("<compartment_ocid>")
  .with_project_id("<project_ocid>")
  .with_subnet_id("<subnet_ocid>")
  .with_shape_name("VM.Standard.E3.Flex")
  .with_shape_config_details(memory_in_gbs=16, ocpus=1) # Applicable only for the
↪ flexible shapes
  .with_block_storage_size(50)
)
.with_runtime(
  ContainerRuntime()
  .with_image("<region>.ocir.io/<your_tenancy>/<your_image>")
  .with_environment_variable(GREETINGS="Welcome to OCI Data Science")
  .with_entrypoint(["/bin/sh", "-c"])
  .with_cmd("sleep 5 && echo $GREETINGS")
)
)

# Create the job with OCI
job.create()
# Run the job and stream the outputs
job_run = job.run().watch()

```

### 15.3.2 YAML

You could use the following YAML to create the same job:

```

kind: job
spec:
  name: container-job
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    spec:
      logGroupId: <log_group_ocid>
      logId: <log_ocid>
      compartmentId: <compartment_ocid>
      projectId: <project_ocid>
      subnetId: <subnet_ocid>
      shapeName: VM.Standard.E3.Flex
      shapeConfigDetails:
        memoryInGBs: 16
        ocpus: 1
        blockStorageSize: 50
      runtime:
        kind: runtime

```

(continues on next page)

(continued from previous page)

```

type: container
spec:
  image: iad.ocir.io/<your_tenancy>/<your_image>
  cmd:
    - sleep 5 && echo $GREETINGS
  entrypoint:
    - /bin/sh
    - -c
  env:
    - name: GREETINGS
      value: Welcome to OCI Data Science

```

### ContainerRuntime Schema

```

kind:
  required: true
  type: string
  allowed:
    - runtime
type:
  required: true
  type: string
  allowed:
    - container
spec:
  type: dict
  required: true
  schema:
    image:
      required: true
      type: string
    entrypoint:
      required: false
      type:
        - string
        - list
    cmd:
      required: false
      type:
        - string
        - list
    env:
      nullable: true
      required: false
      type: list
      schema:
        type: dict
        schema:
          name:
            type: string
          value:
            type:

```

(continues on next page)

(continued from previous page)

- number
- string

## 15.4 Run a Git Repo

The ADS `GitPythonRuntime` class allows you to run source code from a Git repository as a Data Science job. The next example shows how to run a [PyTorch Neural Network Example to train third order polynomial predicting  \$y=\sin\(x\)\$](#) .

### 15.4.1 Python

To configure the `GitPythonRuntime`, you must specify the source code url and entrypoint path. Similar to `PythonRuntime`, you can specify a service conda environment, environment variables, and CLI arguments. In this example, the `pytorch19_p37_gpu_v1` service conda environment is used. Assuming you are running this example in an Data Science notebook session, only log ID and log group ID need to be configured for the `DataScienceJob` object, see [Data Science Jobs](#) for more details about configuring the infrastructure.

```
from ads.jobs import Job, DataScienceJob, GitPythonRuntime

job = (
    Job()
    .with_infrastructure(
        DataScienceJob()
        .with_log_group_id("<log_group_ocid>")
        .with_log_id("<log_ocid>")
        # The following infrastructure configurations are optional
        # if you are in an OCI data science notebook session.
        # The configurations of the notebook session will be used as defaults
        .with_compartment_id("<compartment_ocid>")
        .with_project_id("<project_ocid>")
        .with_subnet_id("<subnet_ocid>")
        .with_shape_name("VM.Standard.E3.Flex")
        .with_shape_config_details(memory_in_gbs=16, ocpus=1) # Applicable only for the
        ↪ flexible shapes
        .with_block_storage_size(50)
    )
    .with_runtime(
        GitPythonRuntime()
        .with_environment_variable(GREETINGS="Welcome to OCI Data Science")
        .with_service_conda("pytorch19_p37_gpu_v1")
        .with_source("https://github.com/pytorch/tutorials.git")
        .with_entrypoint("beginner_source/examples_nn/polynomial_nn.py")
        .with_output(
            output_dir="~/Code/tutorials/beginner_source/examples_nn",
            output_uri="oci://BUCKET_NAME@BUCKET_NAMESPACE/PREFIX"
        )
    )
)

# Create the job with OCI
```

(continues on next page)

(continued from previous page)

```

job.create()
# Run the job and stream the outputs
job_run = job.run().watch()

```

The default branch from the Git repository is used unless you specify a different branch or commit in the `.with_source()` method.

For a public repository, we recommend the “[http://](#)” or “[https://](#)” URL. Authentication may be required for the SSH URL even if the repository is public.

To use a private repository, you must first save an SSH key to an [OCI Vault](#) as a secret, and provide the `secret_ocid` to the `with_source()` method, see [Managing Secret with Vault](#). For example, you could use [GitHub Deploy Key](#).

The entry point specifies how the source code is invoked. The `.with_entrypoint()` has the following arguments:

- `func`: Optional. The function in the script specified by `path` to call. If you don’t specify it, then the script specified by `path` is run as a Python script in a subprocess.
- `path`: Required. The relative path for the script, module, or file to start the job.

With the `GitPythonRuntime` class, you can save the output files from the job run to Object Storage using `with_output()`. By default, the source code is cloned to the `~/Code` directory. In the example, the files in the `example_nn` directory are copied to the Object Storage specified by the `output_uri` parameter. The `output_uri` parameter should have this format:

```
oci://BUCKET_NAME@BUCKET_NAMESPACE/PREFIX
```

The `GitPythonRuntime` also supports these additional configurations:

- The `.with_python_path()` method allows you to add additional Python paths to the runtime. By default, the code directory checked out from Git is added to `sys.path`. Additional Python paths are appended before the code directory is appended.
- The `.with_argument()` method allows you to pass arguments to invoke the script or function. For running a script, the arguments are passed in as CLI arguments. For running a function, the `list` and `dict` JSON serializable objects are supported and are passed into the function.

The `GitPythonRuntime` method updates metadata in the free form tags of the job run after the job run finishes. The following tags are added automatically:

- `commit`: The Git commit ID.
- `method`: The entry function or method.
- `module`: The entry script or module.
- `outputs`: The prefix of the output files in Object Storage.
- `repo`: The URL of the Git repository.

The new values overwrite any existing tags. If you want to skip the metadata update, set `skip_metadata_update` to `True` when initializing the runtime:

```
runtime = GitPythonRuntime(skip_metadata_update=True)
```

## 15.4.2 YAML

You could create the preceding example job with the following YAML file:

```
kind: job
spec:
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    spec:
      logGroupId: <log_group_ocid>
      logId: <log_ocid>
      compartmentId: <compartment_ocid>
      projectId: <project_ocid>
      subnetId: <subnet_ocid>
      shapeName: VM.Standard.E3.Flex
      shapeConfigDetails:
        memoryInGBs: 16
        ocpus: 1
        blockStorageSize: 50
      name: git_example
  runtime:
    kind: runtime
    type: gitPython
    spec:
      entrypoint: beginner_source/examples_nn/polynomial_nn.py
      outputDir: ~/Code/tutorials/beginner_source/examples_nn
      outputUri: oci://BUCKET_NAME@BUCKET_NAMESPACE/PREFIX
      url: https://github.com/pytorch/tutorials.git
      conda:
        slug: pytorch19_p37_gpu_v1
        type: service
      env:
        - name: GREETINGS
          value: Welcome to OCI Data Science
```

### GitPythonRuntime YAML Schema

```
kind:
  required: true
  type: string
  allowed:
    - runtime
type:
  required: true
  type: string
  allowed:
    - gitPython
spec:
  required: true
  type: dict
  schema:
    args:
      type: list
```

(continues on next page)

(continued from previous page)

```

    nullable: true
    required: false
    schema:
      type: string
  branch:
    nullable: true
    required: false
    type: string
  commit:
    nullable: true
    required: false
    type: string
  codeDir:
    required: false
    type: string
  conda:
    nullable: false
    required: false
    type: dict
    schema:
      slug:
        required: true
        type: string
      type:
        required: true
        type: string
        allowed:
          - service
  entryFunction:
    nullable: true
    required: false
    type: string
  entrypoint:
    required: false
    type:
      - string
      - list
  env:
    nullable: true
    required: false
    type: list
    schema:
      type: dict
      schema:
        name:
          type: string
        value:
          type:
            - number
            - string
  outputDir:
    required: false

```

(continues on next page)

(continued from previous page)

```

    type: string
outputUri:
    required: false
    type: string
pythonPath:
    nullable: true
    required: false
    type: list
url:
    required: false
    type: string

```

## 15.5 Run a Notebook

In some cases, you may want to run an existing JupyterLab notebook as a job. You can do this using the `NotebookRuntime()` object.

The next example shows you how to run an the [TensorFlow 2 quick start for beginner](#) notebook from the internet and save the results to OCI Object Storage. The notebook path points to the raw file link from GitHub. To run the following example, ensure that you have internet access to retrieve the notebook:

### 15.5.1 Python

```

from ads.jobs import Job, DataScienceJob, NotebookRuntime

job = (
    Job()
    .with_infrastructure(
        DataScienceJob()
        .with_log_group_id("<log_group_ocid>")
        .with_log_id("<log_ocid>")
        # The following infrastructure configurations are optional
        # if you are in an OCI data science notebook session.
        # The configurations of the notebook session will be used as defaults
        .with_compartment_id("<compartment_ocid>")
        .with_project_id("<project_ocid>")
        .with_subnet_id("<subnet_ocid>")
        .with_shape_name("VM.Standard.E3.Flex")
        .with_shape_config_details(memory_in_gbs=16, ocpus=1) # Applicable only for the_
↪flexible shapes
        .with_block_storage_size(50)
    )
    .with_runtime(
        NotebookRuntime()
        .with_notebook(
            path="https://raw.githubusercontent.com/tensorflow/docs/master/site/en/
↪tutorials/customization/basics.ipynb",
            encoding='utf-8'
        )
    )

```

(continues on next page)



(continued from previous page)

```

        .with_service_conda(tensorflow26_p37_cpu_v2")
        .with_environment_variable(GREETINGS="Welcome to OCI Data Science")
        .with_output("oci://bucket_name@namespace/path/to/dir")
    )
)

job.create()
run = job.run().watch()

```

After the notebook finishes running, the notebook with results are saved to `oci://bucket_name@namespace/path/to/dir`. You can download the output by calling the `download()` method.

```
run.download("/path/to/local/dir")
```

The `NotebookRuntime` also allows you to use exclusion tags, which lets you exclude cells from a job run. For example, you could use these tags to do exploratory data analysis, and then train and evaluate your model in a notebook. Then you could use that same notebook to only build future models that are trained on a different dataset. So the job run only has to execute the cells that are related to training the model, and not the exploratory data analysis or model evaluation.

You tag the cells in the notebook, and then specify the tags using the `.with_exclude_tag()` method. Cells with any matching tags are excluded from the job run. For example, if you tagged cells with `ignore` and `remove`, you can pass in a list of the two tags to the method and those cells are excluded from the code that is executed as part of the job run. To tag cells in a notebook, see [Adding tags using notebook interfaces](#).

```

job.with_runtime(
    NotebookRuntime()
    .with_notebook("path/to/notebook")
    .with_exclude_tag(["ignore", "remove"])
)

```

## 15.5.2 YAML

You could use the following YAML to create the job:

```

kind: job
spec:
  infrastructure:
    kind: infrastructure
  type: dataScienceJob
  spec:
    jobInfrastructureType: STANDALONE
    jobType: DEFAULT
    logGroupId: <log_group_id>
    logId: <log.id>
  runtime:
    kind: runtime
  type: notebook
  spec:
    notebookPathURI: /path/to/notebook
    conda:
      slug: tensorflow26_p37_cpu_v1
      type: service

```

**NotebookRuntime Schema**

```
kind:
  required: true
  type: string
  allowed:
    - runtime
type:
  required: true
  type: string
  allowed:
    - notebook
spec:
  required: true
  type: dict
  schema:
    excludeTags:
      required: false
      type: list
    notebookPathURI:
      required: false
      type: string
    notebookEncoding:
      required: false
      type: string
    outputUri:
      required: false
      type: string
    args:
      nullable: true
      required: false
      type: list
      schema:
        type: string
    conda:
      nullable: false
      required: false
      type: dict
      schema:
        slug:
          required: true
          type: string
        type:
          required: true
          type: string
          allowed:
            - service
    env:
      nullable: true
      required: false
      type: list
      schema:
        type: dict
```

(continues on next page)

(continued from previous page)

```

schema:
name:
  type: string
value:
  type:
    - number
    - string

```

## 15.6 Run a Script

This example shows you how to create a job running “Hello World” Python scripts. Although Python scripts are used here, you could also run Bash or Shell scripts. The Logging service log and log group are defined in the infrastructure. The output of the script appear in the logs.

### 15.6.1 Python

Suppose you would like to run the following “Hello World” python script named `job_script.py`.

```
print("Hello World")
```

First, initiate a job with a job name:

```

from ads.jobs import Job
job = Job(name="Job Name")

```

Next, you specify the desired infrastructure to run the job. If you are in a notebook session, ADS can automatically fetch the infrastructure configurations and use them for the job. If you aren’t in a notebook session or you want to customize the infrastructure, you can specify them using the methods from the `DataScienceJob` class:

```

from ads.jobs import DataScienceJob

job.with_infrastructure(
    DataScienceJob()
    .with_log_group_id("<log_group_ocid>")
    .with_log_id("<log_ocid>")
    # The following infrastructure configurations are optional
    # if you are in an OCI data science notebook session.
    # The configurations of the notebook session will be used as defaults
    .with_compartment_id("<compartment_ocid>")
    .with_project_id("<project_ocid>")
    .with_subnet_id("<subnet_ocid>")
    .with_shape_name("VM.Standard.E3.Flex")
    .with_shape_config_details(memory_in_gbs=16, ocpus=1) # Applicable only for the
    ↪ flexible shapes
    .with_block_storage_size(50)
)

```

In this example, it is a Python script so the `ScriptRuntime()` class is used to define the name of the script using the `.with_source()` method:

```
from ads.jobs import ScriptRuntime
job.with_runtime(
    ScriptRuntime().with_source("job_script.py")
)
```

Finally, you create and run the job, which gives you access to the `job_run.id`:

```
job.create()
job_run = job.run()
```

Additionally, you can acquire the job run using the OCID:

```
from ads.jobs import DataScienceJobRun
job_run = DataScienceJobRun.from_ocid(job_run.id)
```

The `.watch()` method is useful to monitor the progress of the job run:

```
job_run.watch()
```

After the job has been created and runs successfully, you can find the output of the script in the logs if you configured logging.

## 15.6.2 YAML

You could also initialize a job directly from a YAML string. For example, to create a job identical to the preceding example, you could simply run the following:

```
job = Job.from_string(f"""
kind: job
spec:
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    spec:
      logGroupId: <log_group_ocid>
      logId: <log_ocid>
      compartmentId: <compartment_ocid>
      projectId: <project_ocid>
      subnetId: <subnet_ocid>
      shapeName: VM.Standard.E3.Flex
      shapeConfigDetails:
        memoryInGBs: 16
        ocpus: 1
      blockSizeSize: 50
    name: <resource_name>
  runtime:
    kind: runtime
    type: python
    spec:
      scriptPathURI: job_script.py
""")
```

### 15.6.3 Command Line Arguments

If the Python script that you want to run as a job requires CLI arguments, use the `.with_argument()` method to pass the arguments to the job.

#### 15.6.3.1 Python

Suppose you want to run the following python script named `job_script_argument.py`:

```
import sys
print("Hello " + str(sys.argv[1]) + " and " + str(sys.argv[2]))
```

This example runs a job with CLI arguments:

```
job = Job()
job.with_infrastructure(
    DataScienceJob()
    .with_log_id("<log_id>")
    .with_log_group_id("<log_group_id>")
)

# The CLI argument can be passed in using `with_argument` when defining the runtime
job.with_runtime(
    ScriptRuntime()
    .with_source("job_script_argument.py")
    .with_argument("<first_argument>", "<second_argument>")
)

job.create()
job_run = job.run()
```

After the job run is created and run, you can use the `.watch()` method to monitor its progress:

```
job_run.watch()
```

This job run prints out Hello `<first_argument>` and `<second_argument>`.

#### 15.6.3.2 YAML

You could create the preceding example job with the following YAML file:

```
kind: job
spec:
  infrastructure:
kind: infrastructure
type: dataScienceJob
spec:
  logGroupId: <log_group_ocid>
  logId: <log_ocid>
  compartmentId: <compartment_ocid>
  projectId: <project_ocid>
  subnetId: <subnet_ocid>
```

(continues on next page)

(continued from previous page)

```

shapeName: VM.Standard.E3.Flex
shapeConfigDetails:
  memoryInGBs: 16
  ocpus: 1
blockStorageSize: 50
runtime:
  kind: runtime
type: python
spec:
  args:
    - <first_argument>
    - <second_argument>
  scriptPathURI: job_script_argument.py

```

## 15.6.4 Environment Variables

Similarly, if the script you want to run requires environment variables, you also pass them in using the `.with_environment_variable()` method. The key-value pair of the environment variable are passed in using the `.with_environment_variable()` method, and are accessed in the Python script using the `os.environ` dictionary.

### 15.6.4.1 Python

Suppose you want to run the following python script named `job_script_env.py`:

```

import os
import sys
print("Hello " + os.environ["KEY1"] + " and " + os.environ["KEY2"])

```

This example runs a job with environment variables:

```

job = Job()
job.with_infrastructure(
    DataScienceJob()
    .with_log_group_id("<log_group_ocid>")
    .with_log_id("<log_ocid>")
    # The following infrastructure configurations are optional
    # if you are in an OCI data science notebook session.
    # The configurations of the notebook session will be used as defaults
    .with_compartment_id("<compartment_ocid>")
    .with_project_id("<project_ocid>")
    .with_subnet_id("<subnet_ocid>")
    .with_shape_name("VM.Standard.E3.Flex")
    .with_shape_config_details(memory_in_gbs=16, ocpus=1)
    .with_block_storage_size(50)
)

job.with_runtime(
    ScriptRuntime()
    .with_source("job_script_env.py")
    .with_environment_variable(KEY1="<first_value>", KEY2="<second_value>")
)

```

(continues on next page)

(continued from previous page)

```
)
job.create()
job_run = job.run()
```

You can watch the progress of the job run using the `.watch()` method:

```
job_run.watch()
```

This job run prints out Hello `<first_value>` and `<second_value>`.

#### 15.6.4.2 YAML

You could create the preceding example job with the following YAML file:

```
kind: job
spec:
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    spec:
      logGroupId: <log_group_ocid>
      logId: <log_ocid>
      compartmentId: <compartment_ocid>
      projectId: <project_ocid>
      subnetId: <subnet_ocid>
      shapeName: VM.Standard.E3.Flex
      shapeConfigDetails:
        memoryInGBs: 16
        ocpus: 1
        blockStorageSize: 50
  runtime:
    kind: runtime
    type: python
    spec:
      env:
        - name: KEY1
          value: <first_value>
        - name: KEY2
          value: <second_value>
      scriptPathURI: job_script_env.py
```

#### ScriptRuntime YAML Schema

```
kind:
  required: true
  type: string
  allowed:
    - runtime
type:
  required: true
  type: string
  allowed:
```

(continues on next page)

(continued from previous page)

```
- script
spec:
  required: true
  type: dict
  schema:
    args:
      nullable: true
      required: false
      type: list
      schema:
        type: string
    conda:
      nullable: false
      required: false
      type: dict
      schema:
        slug:
          required: true
          type: string
        type:
          allowed:
            - service
          required: true
          type: string
    env:
      nullable: true
      required: false
      type: list
      schema:
        type: dict
        schema:
          name:
            type: string
          value:
            type:
              - number
              - string
    scriptPathURI:
      required: true
      type: string
    entrypoint:
      required: false
      type: string
```



## 15.7 Run Code in ZIP or Folder

### 15.7.1 ScriptRuntime

The `ScriptRuntime` class is designed for you to define job artifacts and configurations supported by OCI Data Science jobs natively. It can be used with any script types that is supported by the OCI Data Science jobs, including a ZIP or compressed tar file or folder. See [Preparing Job Artifacts](#) for more details. In the job run, the working directory is the user's home directory. For example `/home/datascience`.

#### 15.7.1.1 Python

If you are in a notebook session, ADS can automatically fetch the infrastructure configurations, and use them in the job. If you aren't in a notebook session or you want to customize the infrastructure, you can specify them using the methods in the `DataScienceJob` class.

With the `ScriptRuntime`, you can pass in a path to a ZIP file or directory. For a ZIP file, the path can be any URI supported by `fsspec`, including OCI Object Storage.

You must specify the `entrypoint`, which is the relative path from the ZIP file or directory to the script starting your program. Note that the `entrypoint` contains the name of the directory, since the directory itself is also zipped as the job artifact.

```
from ads.jobs import Job, DataScienceJob, ScriptRuntime

job = (
    Job()
    .with_infrastructure(
        DataScienceJob()
        .with_log_group_id("<log_group_ocid>")
        .with_log_id("<log_ocid>")
        # The following infrastructure configurations are optional
        # if you are in an OCI data science notebook session.
        # The configurations of the notebook session will be used as defaults
        .with_compartment_id("<compartment_ocid>")
        .with_project_id("<project_ocid>")
        .with_subnet_id("<subnet_ocid>")
        .with_shape_name("VM.Standard.E3.Flex")
        .with_shape_config_details(memory_in_gbs=16, ocpus=1)
        .with_block_storage_size(50)
    )
    .with_runtime(
        ScriptRuntime()
        .with_source("path/to/zip_or_dir", entrypoint="zip_or_dir/main.py")
        .with_service_conda("pytorch19_p37_cpu_v1")
    )
)

# Create the job with OCI
job.create()
# Run the job and stream the outputs
job_run = job.run().watch()
```

### 15.7.1.2 YAML

You could use the following YAML example to create the same job with `ScriptRuntime`:

```
kind: job
spec:
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    spec:
      logGroupId: <log_group_ocid>
      logId: <log_ocid>
      compartmentId: <compartment_ocid>
      projectId: <project_ocid>
      subnetId: <subnet_ocid>
      shapeName: VM.Standard.E3.Flex
      shapeConfigDetails:
        memoryInGBs: 16
        ocpus: 1
        blockStorageSize: 50
  runtime:
    kind: runtime
    type: script
    spec:
      conda:
        slug: pytorch19_p37_cpu_v1
        type: service
      entrypoint: zip_or_dir/main.py
      scriptPathURI: path/to/zip_or_dir
```

## 15.7.2 PythonRuntime

The `PythonRuntime` class allows you to run Python code with ADS enhanced features like configuring the working directory and Python path. It also allows you to copy the output files to OCI Object Storage. This is especially useful for Python code involving multiple files and packages in the job artifact.

The `PythonRuntime` uses an ADS generated driver script as the entry point for the job run. It performs additional operations before and after invoking your code. You can examine the driver script by downloading the job artifact from the OCI Console.

### 15.7.2.1 Python

Relative to `ScriptRunTime` the `PythonRuntime` has 3 additional methods:

- `.with_working_dir()`: Specify the working directory to use when running a job. By default, the working directory is also added to the Python paths. This should be a relative path from the parent of the job artifact directory.
- `.with_python_path()`: Add one or more Python paths to use when running a job. The paths should be relative paths from the working directory.
- `.with_output()`: Specify the output directory and a remote URI (for example, an OCI Object Storage URI) in the job run. Files in the output directory are copied to the remote output URI after the job run finishes successfully.

Following is an example of creating a job with `PythonRuntime`:

```

from ads.jobs import Job, DataScienceJob, PythonRuntime

job = (
    Job()
    .with_infrastructure(
        DataScienceJob()
        .with_log_group_id("<log_group_ocid>")
        .with_log_id("<log_ocid>")
        # The following infrastructure configurations are optional
        # if you are in an OCI data science notebook session.
        # The configurations of the notebook session will be used as defaults
        .with_compartment_id("<compartment_ocid>")
        .with_project_id("<project_ocid>")
        .with_subnet_id("<subnet_ocid>")
        .with_shape_name("VM.Standard.E3.Flex")
        .with_shape_config_details(memory_in_gbs=16, ocpus=1) # Applicable only for the
↪flexible shapes
        .with_block_storage_size(50)
    )
    .with_runtime(
        PythonRuntime()
        .with_service_conda("pytorch19_p37_cpu_v1")
        # The job artifact directory is named "zip_or_dir"
        .with_source("local/path/to/zip_or_dir", entrypoint="zip_or_dir/my_package/entry.py")
        # Change the working directory to be inside the job artifact directory
        # Working directory a relative path from the parent of the job artifact directory
        # Working directory is also added to Python paths
        .with_working_dir("zip_or_dir")
        # Add an additional Python path
        # The "my_python_packages" folder is under "zip_or_dir" (working directory)
        .with_python_path("my_python_packages")
        # Files in "output" directory will be copied to OCI object storage once the job
↪finishes
        # Here we assume "output" is a folder under "zip_or_dir" (working directory)
        .with_output("output", "oci://bucket_name@namespace/path/to/dir")
    )
)

```

### 15.7.2.2 YAML

You could use the following YAML to create the same job with PythonRuntime:

```

kind: job
spec:
  infrastructure:
    kind: infrastructure
    type: dataScienceJob
    spec:
      logGroupId: <log_group_ocid>
      logId: <log_ocid>
      compartmentId: <compartment_ocid>
      projectId: <project_ocid>

```

(continues on next page)

(continued from previous page)

```

    subnetId: <subnet_ocid>
    shapeName: VM.Standard.E3.Flex
    shapeConfigDetails:
      memoryInGBs: 16
      ocpus: 1
      blockStorageSize: 50
  runtime:
    kind: runtime
    type: python
    spec:
      conda:
        slug: pytorch19_p37_cpu_v1
        type: service
      entrypoint: zip_or_dir/my_package/entry.py
      scriptPathURI: path/to/zip_or_dir
      workingDir: zip_or_dir
      outputDir: zip_or_dir/output
      outputUri: oci://bucket_name@namespace/path/to/dir
      pythonPath:
        - "zip_or_dir/python_path"

```

### PythonRuntime YAML Schema

```

kind:
  required: true
  type: string
  allowed:
    - runtime
type:
  required: true
  type: string
  allowed:
    - script
spec:
  required: true
  type: dict
  schema:
    args:
      nullable: true
      required: false
      type: list
      schema:
        type: string
    conda:
      nullable: false
      required: false
      type: dict
      schema:
        slug:
          required: true
          type: string
        type:

```

(continues on next page)

(continued from previous page)

```
    allowed:
      - service
    required: true
    type: string
env:
  nullable: true
  required: false
  type: list
  schema:
    type: dict
    schema:
      name:
        type: string
      value:
        type:
          - number
          - string
scriptPathURI:
  required: true
  type: string
entrypoint:
  required: false
  type: string
outputDir:
  required: false
  type: string
outputUri:
  required: false
  type: string
workingDir:
  required: false
  type: string
pythonPath:
  required: false
  type: list
```

## 15.8 Working with OCI Data Science Jobs Using CLI

### 15.8.1 Prerequisite

- Complete *Build Development Container Image*

## 15.8.2 Running a Pre Defined Job

```
aads opctl run -j <job ocid>
```

## 15.8.3 Delete Job or Job Run

```
ads opctl delete <job-id or run-id>
```

## 15.8.4 Cancel Job Run

```
ads opctl cancel <run-id>
```

## 15.8.5 Cancel Distributed Training Job

Stop a running cluster using `cancel` subcommand.

### Option 1: Using Job OCID and Work Dir

```
ads opctl cancel -j <job ocid> --work-dir <Object storage working directory specified,  
↪when the cluster was created>
```

### Option 2: Using cluster info file

Cluster info file is a yaml file with output generated from `ads opctl run -f`

```
ads opctl cancel -j <job ocid> --work-dir <Object storage working directory specified,  
↪when the cluster was created>
```

This command requires an api key or resource principal setup. The logs are streamed from the logging service. If your job is not attached to logging service, this option will show only the lifecycle state.

## 15.9 Monitoring With CLI

### 15.9.1 watch

You can tail the logs generated by OCI Data Science Job Runs or OCI DataFlow Application Runs using the `watch` subcommand.

```
ads opctl watch <job run ocid or dataflow application run ocid>
```

This command requires an api key or resource principal setup. The logs are streamed from the logging service. If your job is not attached to logging service, this option will show only the lifecycle state.

## STORE CREDENTIALS

Services such as OCI Database and Streaming require users to provide credentials. These credentials must be safely accessed at runtime. [OCI Vault](#) provides a mechanism for safe storage and access of secrets. `SecretKeeper` uses Vault as a backend to store and retrieve the credentials. The data structure of the credentials varies from service to service. There is a `SecretKeeper` specific to each data structure.

These classes are provided:

- `ADBSecretKeeper`: Stores credentials for the Oracle Autonomous Database, with or without the wallet file.
- `AuthTokenSecretKeeper`: Stores an Auth Token or Access Token string. This could be an Auth Token to use to connect to Streaming, Github, or other systems that used Auth Tokens or Access Token strings.
- `BDSecretKeeper`: Stores credentials for Oracle Big Data Service with or without Keytab and kerb5 configuration files.
- `MySQLDBSecretKeeper`: Stores credentials for the MySQL database. This class will work with many databases that authenticate with a username and password only.
- `OracleDBSecretKeeper`: Stores credentials for the Oracle Database.

### 16.1 Quick Start

#### 16.1.1 Auth Tokens

##### 16.1.1.1 Save Credentials

```
import ads
from ads.secrets.auth_token import AuthTokenSecretKeeper

ads.set_auth('resource_principal') # If using resource principal authentication

ocid_vault = "ocid1.vault..<unique_ID>"
ocid_master_key = "ocid1.key..<unique_ID>"
ocid_mycompartment = "ocid1.compartment..<unique_ID>"

authtoken2 = AuthTokenSecretKeeper(
    vault_id=ocid_vault,
    key_id=ocid_master_key,
    compartment_id=ocid_mycompartment,
    auth_token="<your_auth_token>"
).save()
```

(continues on next page)

(continued from previous page)

```

        "my_xyz_auth_token2",
        "This is my key for git repo xyz",
        freeform_tags={"gitrepo":"xyz"}
    )
print(authtoken2.secret_id)

```

```
'ocid1.vaultsecret..<unique_ID>'
```

### 16.1.1.2 Load Credentials

```

import ads
from ads.secrets.auth_token import AuthTokenSecretKeeper

ads.set_auth('resource_principal') # If using resource principal authentication

with AuthTokenSecretKeeper.load_secret(source="ocid1.vaultsecret..<unique_ID>",
                                       ) as authtoken:
    import os
    print(f"Credentials inside `authtoken` object: {authtoken}")

```

```
Credentials inside `authtoken` object: {'auth_token': '<your_auth_token>'}
```

## 16.1.2 Autonomous Database

### 16.1.2.1 Save Credentials

```

import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.adb import ADBSecretKeeper

connection_parameters={
    "user_name":"admin",
    "password":"<your_password>",
    "service_name":"service_high",
    "wallet_location":"/home/datascience/Wallet_-----.zip"
}

ocid_vault = "ocid1.vault..<unique_ID>"
ocid_master_key = "ocid1.key..<unique_ID>"
ocid_mycompartment = "ocid1.compartment..<unique_ID>"

adw_keeper = ADBSecretKeeper(vault_id=ocid_vault,
                             key_id=ocid_master_key,
                             compartment_id=ocid_mycompartment,
                             **connection_parameters)

# Store the credentials without storing the wallet file
adw_keeper.save("adw_employee_att2",
               "My DB credentials",
               freeform_tags={"schema":"emp"},

```

(continues on next page)



(continued from previous page)

```

        save_wallet=True
    )
print(adw_keeper.secret_id)

```

```
'ocid1.vaultsecret..<unique_ID>'
```

### 16.1.2.2 Load Credentials

```

import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.adb import ADBSecretKeeper

with ADBSecretKeeper.load_secret("ocid1.vaultsecret..<unique_ID>") as adw_creds2:
    import pandas as pd
    df2 = pd.DataFrame(ads.read_sql("select JOBFUNCTION, ATTRITION from ATTRITION_DATA",
    ↪connection_parameters=adw_creds2))
    print(df2.head(2))

```

	JOBFUNCTION	ATTRITION
0	Product Management	No
1	Software Developer	No

## 16.1.3 Big Data Service

### 16.1.3.1 Save Credentials

```

import ads
import fsspec
import os

from ads.secrets.big_data_service import BDSSecretKeeper
from ads.bds.auth import has_kerberos_ticket, refresh_ticket, krbcontext

ads.set_auth('resource_principal')

principal = "<your_principal>"
hdfs_host = "<your_hdfs_host>"
hive_host = "<your_hive_host>"
hdfs_port = <your_hdfs_port>
hive_port = <your_hive_port>
vault_id = "ocid1.vault..<unique_ID>"
key_id = "ocid1.key..<unique_ID>"

secret = BDSSecretKeeper(
    vault_id=vault_id,
    key_id=key_id,
    principal=principal,
    hdfs_host=hdfs_host,

```

(continues on next page)

(continued from previous page)

```

        hive_host=hive_host,
        hdfs_port=hdfs_port,
        hive_port=hive_port,
        keytab_path=keytab_path,
        kerb5_path=kerb5_path
    )

saved_secret = secret.save(name="your_bds_config_secret_name",
                           description="your bds credentials",
                           freeform_tags={"schema":"emp"},
                           defined_tags={},
                           save_files=True)

```

### 16.1.3.2 Load Credentials

```

from ads.secrets.big_data_service import BDSSecretKeeper
from pyhive import hive

with BDSSecretKeeper.load_secret(saved_secret.secret_id, keytab_dir="~/path/to/save/
↳keytab_file/") as cred:
    with krbcontext(principal=cred["principal"], keytab_path=cred['keytab_path']):
        hive_cursor = hive.connect(host=cred["hive_host"],
                                    port=cred["hive_port"],
                                    auth='KERBEROS',
                                    kerberos_service_name="hive").cursor()

```

## 16.1.4 MySQL

### 16.1.4.1 Save Credentials

```

import ads
from ads.secrets.mysqlldb import MySQLDBSecretKeeper

vault_id = "ocid1.vault.<unique_ID>"
key_id = "ocid1.key.<unique_ID>"

ads.set_auth("resource_principal") # If using resource principal for authentication
connection_parameters={
    "user_name": "<your user name>",
    "password": "<your password>",
    "host": "<db host>",
    "port": "<db port>",
    "database": "<database>",
}

mysqlldb_keeper = MySQLDBSecretKeeper(vault_id=vault_id,
                                       key_id=key_id,
                                       **connection_parameters)

```

(continues on next page)

(continued from previous page)

```
mysqldb_keeper.save("mysqldb_employee", "My DB credentials", freeform_tags={"schema":"emp
↪"})
print(mysqldb_keeper.secret_id) # Prints the secret_id of the stored credentials
```

```
'ocid1.vaultsecret..<unique_ID>'
```

### 16.1.4.2 Load Credentials

```
import ads
from ads.secrets.mysqldb import MySQLDBSecretKeeper
ads.set_auth('resource_principal') # If using resource principal authentication

with MySQLDBSecretKeeper.load_secret(source=secret_id) as mysqldb_creds:
    import pandas as pd
    df2 = pd.DataFrame(ads.read_sql("select JOBFUNCTION, ATTRITION from ATTRITION_DATA",
↪connection_parameters=mysqldb_creds)
    print(df2.head(2))
```

	JOBFUNCTION	ATTRITION
0	Product Management	No
1	Software Developer	No

## 16.1.5 Oracle Database

### 16.1.5.1 Save Credentials

```
import ads
from ads.secrets.oracledb import OracleDBSecretKeeper

vault_id = "ocid1.vault..<unique_ID>"
key_id = "ocid1.key..<unique_ID>"

ads.set_auth("resource_principal") # If using resource principal for authentication
connection_parameters={
    "user_name": "<your user name>",
    "password": "<your password>",
    "service_name": "service_name",
    "host": "<db host>",
    "port": "<db port>",
}

oracledb_keeper = OracleDBSecretKeeper(vault_id=vault_id,
                                       key_id=key_id,
                                       **connection_parameters)

oracledb_keeper.save("oracledb_employee", "My DB credentials", freeform_tags={"schema":
↪"emp"})
print(oracledb_keeper.secret_id) # Prints the secret_id of the stored credentials
```

```
'ocid1.vaultsecret..<unique_ID>'
```

### 16.1.5.2 Load Credentials

```
import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.oracledb import OracleDBSecretKeeper

with OracleDBSecretKeeper.load_secret(source=secret_id) as oracledb_creds:
    import pandas as pd
    df2 = pd.DataFrame(ads.read_sql("select JOBFUNCTION, ATTRITION from ATTRITION_DATA",
    ↪connection_parameters=oracledb_creds)
    print(df2.head(2))
```

	JOBFUNCTION	ATTRITION
0	Product Management	No
1	Software Developer	No

## 16.2 Auth Token

The `AuthTokenSecretKeeper` helps you to save the Auth Token or Access Token string to the OCI Vault service.

See [API Documentation](#) for more details

### 16.2.1 Save Credentials

#### 16.2.1.1 AuthTokenSecretKeeper

The `AuthTokenSecretKeeper` constructor takes the following parameters:

- `auth_token` (str): Provide the Auth Token or Access Token string to be stored
- `vault_id` (str): ocid of the vault
- `key_id` (str): ocid of the master key used for encrypting the secret
- `compartment_id` (str, optional): Default is None. ocid of the compartment where the vault is located. This will be defaulted to the compartment of the Notebook session, if used within a OCI Data Science notebook session.

##### 16.2.1.1.1 Save

The `AuthTokenSecretKeeper.save` API serializes and stores the credentials to Vault. It takes following parameters -

- `name` (str): Name of the secret when saved in the vault.
- `description` (str): Description of the secret when saved in the vault.
- `freeform_tags` (dict, optional): Freeform tags to use when saving the secret in the OCI Console.
- `defined_tags` (dict, optional.): Save the tags under predefined tags in the OCI Console.

The secret has following information:

- auth\_token

### 16.2.1.2 Examples

#### 16.2.1.2.1 Save Auth Token

```
import ads
from ads.secrets.auth_token import AuthTokenSecretKeeper

ads.set_auth('resource_principal') # If using resource principal authentication

ocid_vault = "ocid1.vault...<unique_ID>"
ocid_master_key = "ocid1.key...<unique_ID>"
ocid_mycompartment = "ocid1.compartment...<unique_ID>"

authtoken2 = AuthTokenSecretKeeper(
    vault_id=ocid_vault,
    key_id=ocid_master_key,
    compartment_id=ocid_mycompartment,
    auth_token="<your_auth_token>"
).save(
    "my_xyz_auth_token2",
    "This is my key for git repo xyz",
    freeform_tags={"gitrepo":"xyz"}
)
print(authtoken2.secret_id)
```

You can save the vault details in a file for later reference or using it within your code using `export_vault_details` API. The API currently let us export the information as a yaml file or a json file.

```
authtoken2.export_vault_details("my_db_vault_info.json", format="json")
```

#### 16.2.1.2.2 Save as a yaml File

```
authtoken2.export_vault_details("my_db_vault_info.yaml", format="yaml")
```

## 16.2.2 Load Credentials

### 16.2.2.1 Load

The `AuthTokenSecretKeeper.load_secret` API deserializes and loads the credentials from Vault. You could use this API in one of the following ways:

### 16.2.2.1.1 Using a with Statement

```
with AuthTokenSecretKeeper.load_secret('ocidl.vaultsecret..<unique_ID>') as authtoken:
    print(authtoken['user_name'])
```

This approach is preferred as the secrets are only available within the code block and it reduces the risk that the variable will be leaked.

### 16.2.2.1.2 Without using a with Statement

```
authtoken = AuthTokenSecretKeeper.load_secret('ocidl.vaultsecret..<unique_ID>')
authtokendict = authtoken.to_dict()
print(authtokendict['user_name'])
```

The `.load_secret()` takes the following parameters:

- `auth`: Provide overriding authorization information if the authorization information is different from the `ads.set_auth` setting.
- `export_env`: Default is `False`. If set to `True`, the credentials are exported as environment variable when used with
- `export_prefix`: The default name for environment variable is `user_name`, `password`, `service_name`, and `wallet_location`. You can add a prefix to avoid name collision
- `format`: Optional. If source is a file, then this value must be `json` or `yaml` depending on the file format.
- `source`: Either the file that was exported from `export_vault_details` or the OCID of the secret
- the `with` operator.

## 16.2.2.2 Examples

### 16.2.2.2.1 Using a with Statement

```
import ads
from ads.secrets.auth_token import AuthTokenSecretKeeper

ads.set_auth('resource_principal') # If using resource principal authentication

with AuthTokenSecretKeeper.load_secret(source="ocidl.vaultsecret..<unique_ID",
                                       ) as authtoken:
    import os
    print(f"Credentials inside `authtoken` object: {authtoken}")
```

Credentials inside `authtoken` object: {'auth\_token': '<your\_auth\_token>'}

### 16.2.2.2.2 Export to Environment Variables Using a with Statement

To expose credentials through environment variable, set `export_env=True`. The following keys are exported -

Secret attribute	Environment Variable Name
auth_token	auth_token

```
import ads
from ads.secrets.auth_token import AuthTokenSecretKeeper
import os

ads.set_auth('resource_principal') # If using resource principal authentication

with AuthTokenSecretKeeper.load_secret(
    source="ocid1.vaultsecret.<unique_ID>",
    export_env=True
):
    print(os.environ.get("auth_token")) # Prints the auth token

print(os.environ.get("auth_token")) # Prints nothing. The credentials are cleared from
↳ the dictionary outside the ``with`` block
```

You can avoid name collisions by setting the prefix string using `export_prefix` along with `export_env=True`. For example, if you set the prefix to `kafka`, the exported keys are:

Secret attribute	Environment Variable Name
auth_token	kafka.auth_token

```
import ads
from ads.secrets.auth_token import AuthTokenSecretKeeper
import os

ads.set_auth('resource_principal') # If using resource principal authentication

with AuthTokenSecretKeeper.load_secret(
    source="ocid1.vaultsecret.<unique_ID>",
    export_env=True,
    export_prefix="kafka"
):
    print(os.environ.get("kafka.auth_token")) # Prints the auth token

print(os.environ.get("kafka.auth_token")) # Prints nothing. The credentials are cleared
↳ from the dictionary outside the ``with`` block
```

## 16.3 Autonomous Database

To connect to Autonomous Database you need the following:

- user name
- password
- service name
- [wallet file](#)

The `ADBSecretKeeper` class saves the ADB credentials to the OCI Vault service.

See [API Documentation](#) for more details

### 16.3.1 Save Credentials

#### 16.3.1.1 `ADBSecretKeeper`

The `ADBSecretKeeper` constructor has the following parameters:

- `compartment_id` (str): OCID of the compartment where the vault is located. This defaults to the compartment of the notebook session when used in a Data Science notebook session.
- `key_id` (str): OCID of the master key used for encrypting the secret.
- `password` (str): The password of the database.
- `service_name` (str): Set the service name of the database.
- `user_name` (str): The user name to be stored.
- `vault_id` (str): OCID of the vault.
- `wallet_location` (str): Path to the wallet ZIP file.

##### 16.3.1.1.1 Save

The `ADBSecretKeeper.save` API serializes and stores the credentials to Vault using the following parameters:

- `defined_tags` (dict, optional): Default None. Save the tags under predefined tags in the OCI Console.
- `description` (str): Description of the secret when saved in Vault.
- `freeform_tags` (dict, optional): Default None. Free form tags to use for saving the secret in the OCI Console.
- `name` (str): Name of the secret when saved in Vault.
- `save_wallet` (bool, optional): Default False. If set to True, then the wallet file is serialized.

When stored without the wallet information, the secret content has following information:

- `password`
- `service_name`
- `user_name`

To store wallet file content, set `save_wallet` to True. The wallet content is stored by extracting all the files from the wallet ZIP file, and then each file is stored in the vault as a secret. The list of OCIDs corresponding to each file along with username, password, and service name is stored in a separate secret. The secret corresponding to each file content has following information:



- filename
- content of the file

A **meta secret** is created to save the username, password, service name, and the secret ids of the files within the wallet file. It has following attributes:

- user\_name
- password
- wallet\_file\_name
- wallet\_secret\_ids

The wallet file is reconstructed when `ADBSecretKeeper.load_secret` is called using the OCID of the **meta secret**.

### 16.3.1.2 Examples

#### 16.3.1.2.1 Without the Wallet File

```
import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.adb import ADBSecretKeeper

connection_parameters={
    "user_name":"admin",
    "password":"<your_password>",
    "service_name":"service_high",
    "wallet_location":"/home/datascience/Wallet_-----.zip"
}

ocid_vault = "ocid1.vault.<unique_ID>"
ocid_master_key = "ocid1.key.<unique_ID>"
ocid_mycompartment = "ocid1.compartment.<unique_ID>"

adw_keeper = ADBSecretKeeper(vault_id=ocid_vault,
                             key_id=ocid_master_key,
                             compartment_id=ocid_mycompartment,
                             **connection_parameters)

# Store the credentials without storing the wallet file
adw_keeper.save("adw_employee_att2", "My DB credentials", freeform_tags={"schema":"emp"})
print(adw_keeper.secret_id)
```

```
'ocid1.vaultsecret.<unique_ID>'
```

### 16.3.1.2.2 With the Wallet File

```
import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.adb import ADBSecretKeeper

connection_parameters={
    "user_name":"admin",
    "password":"<your_password>",
    "service_name":"service_high",
    "wallet_location":"/home/datascience/Wallet_-----.zip"
}

ocid_vault = "ocid1.vault.<unique_ID>"
ocid_master_key = "ocid1.key.<unique_ID>"
ocid_mycompartment = "ocid1.compartment.<unique_ID>"

adw_keeper = ADBSecretKeeper(vault_id=ocid_vault,
                             key_id=ocid_master_key,
                             compartment_id=ocid_mycompartment,
                             **connection_parameters)

# Set `save_wallet`=True to save wallet file

adw_keeper.save("adw_employee_att2",
               "My DB credentials",
               freeform_tags={"schema":"emp"},
               save_wallet=True
)

print(adw_keeper.secret_id)
```

'ocid1.vaultsecret.<unique\_ID>'

You can save the vault details in a file for later reference or using it within your code using `export_vault_details` API calls. The API currently enables you to export the information as a YAML file or a JSON file.

```
adw_keeper.export_vault_details("my_db_vault_info.json", format="json")
```

To save as a YAML file:

```
adw_keeper.export_vault_details("my_db_vault_info.yaml", format="yaml")
```

## 16.3.2 Load Credentials

### 16.3.2.1 Load

The `ADBSecretKeeper.load_secret` API deserializes and loads the credentials from Vault. You could use this API in one of the following ways:

### 16.3.2.1.1 Using a with Statement

```
with ADBSecretKeeper.load_secret('ocid1.vaultsecret..<unique_ID>') as adwsecret:
    print(adwsecret['user_name'])
```

This approach is preferred as the secrets are only available within the code block and it reduces the risk that the variable will be leaked.

### 16.3.2.1.2 Without using a with Statement

```
adwsecretobj = ADBSecretKeeper.load_secret('ocid1.vaultsecret..<unique_ID>')
adwsecret = adwsecretobj.to_dict()
print(adwsecret['user_name'])
```

The `.load_secret()` method has the following parameters:

- **auth:** Provide overriding authorization information if the authorization information is different from the `ads.set_auth` setting.
- **export\_env:** Default is `False`. If set to `True`, the credentials are exported as environment variable when used with the `with` operator.
- **export\_prefix:** The default name for environment variable is `user_name`, `password`, `service_name`, and `wallet_location`. You can add a prefix to avoid name collision
- **format:** Optional. If `source` is a file, then this value must be `json` or `yaml` depending on the file format.
- **source:** Either the file that was exported from `export_vault_details` or the OCID of the secret
- **wallet\_dir:** Optional. Directory path where the wallet zip file will be saved after the contents are retrieved from Vault. If wallet content is not available in the provided secret OCID, this attribute is ignored.
- **wallet\_location:** Optional. Path to the local wallet zip file. If vault secret does not have wallet file content, set this variable so that it will be available in the exported credential. If provided, this path takes precedence over the wallet file information in the secret.

If the wallet file was saved in the vault, then the ZIP file of the same name is created by the `.load_secret()` method. By default the ZIP file is created in the working directory. To update the location, you can set the directory path with `wallet_dir`.

## 16.3.2.2 Examples

### 16.3.2.2.1 Using a with Statement

```
import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.adb import ADBSecretKeeper

with ADBSecretKeeper.load_secret(
    "ocid1.vaultsecret..<unique_ID>"
) as adw_creds2:
    print(adw_creds2["user_name"]) # Prints the user name
```

(continues on next page)

(continued from previous page)

```
print (adw_creds2["user_name"]) # Prints nothing. The credentials are cleared from the
↪dictionary outside the ``with`` block
```

### 16.3.2.2.2 Export to Environment Variables Using a with Statement

To expose credentials as an environment variable, set `export_env=True`. The following keys are exported:

Secret attribute	Environment Variable Name
user_name	user_name
password	password
service_name	service_name
wallet_location	wallet_location

```
import os
import ads

ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.adb import ADBSecretKeeper

with ADBSecretKeeper.load_secret(
    "ocid1.vaultsecret..<unique_ID>",
    export_env=True
):
    print(os.environ.get("user_name")) # Prints the user name

print(os.environ.get("user_name")) # Prints nothing. The credentials are cleared from
↪the dictionary outside the ``with`` block
```

You can avoid name collisions by setting a prefix string using `export_prefix` along with `export_env=True`. For example, if you set the prefix to `myprocess`, then the exported keys are:

Secret attribute	Environment Variable Name
user_name	myprocess.user_name
password	myprocess.password
service_name	myprocess.service_name
wallet_location	myprocess.wallet_location

```
import os
import ads

ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.adb import ADBSecretKeeper

with ADBSecretKeeper.load_secret(
    "ocid1.vaultsecret..<unique_ID>",
    export_env=True,
    export_prefix="myprocess"
):
    print(os.environ.get("myprocess.user_name")) # Prints the user name
```

(continues on next page)

(continued from previous page)

```
print(os.environ.get("myprocess.user_name")) # Prints nothing. The credentials are_
↳cleared from the dictionary outside the ``with`` block
```

### 16.3.2.2.3 Wallet File Location

You can set wallet file location when wallet file is not part of the stored vault secret. To specify a local wallet ZIP file, set the path to the ZIP file with `wallet_location`:

```
import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.adb import ADBSecretKeeper

with ADBSecretKeeper.load_secret(
    "ocid1.vaultsecret.<unique_ID>",
    wallet_location="path/to/my/local/wallet.zip"
) as adw_creds2:
    print (adw_creds2["wallet_location"]) # Prints `path/to/my/local/wallet.zip`

print (adw_creds2["wallet_location"]) # Prints nothing. The credentials are cleared from_
↳the dictionary outside the ``with`` block
```

## 16.4 Big Data Service

New in version 2.5.10..

To connect to Oracle Big Data Service (BDS) you need the following:

- `hdfs host`: HDFS hostname which will be used to connect to the HDFS file system.
- `hdfs port`: HDFS port which will be used to connect to the HDFS file system.
- `hive host`: Hive hostname which will be used to connect to the Hive Server.
- `hive port`: Hive port which will be used to connect to the Hive Server.
- `kerb5 config file`: `krb5.conf` file which can be copied from `/etc/krb5.conf` from the master node of the BDS cluster. It will be used to generate the kerberos ticket.
- `keytab file`: The principal's keytab file which can be downloaded from the master node of the BDS cluster. It will be used to generate the kerberos ticket.
- `principal`: The unique identity to which Kerberos can assign tickets. It will be used to generate the kerberos ticket.

The `BDSSecretKeeper` class saves the BDS credentials to the OCI Vault service.

See [API Documentation](#) for more details

## 16.4.1 Save Credentials

### 16.4.1.1 BDSSecretKeeper

You can also save the connection parameters as well as the files needed to configure the kerberos authentication into vault. This will allow you to use repetitively in different notebook sessions, machines, and Jobs.

The `BDSSecretKeeper` constructor requires the following parameters:

- `compartment_id` (str): OCID of the compartment where the vault is located. This defaults to the compartment of the notebook session when used in a Data Science notebook session.
- `hdfs_host` (str): The HDFS hostname from the bds cluster.
- `hdfs_port` (str): The HDFS port from the bds cluster.
- `hive_host` (str): The Hive hostname from the bds cluster.
- `hive_port` (str): The Hive port from the bds cluster.
- `kerb5_path` (str): The `krb5.conf` file path.
- `key_id`: str (OCID of the master key used for encrypting the secret).
- `keytab_path` (str): The path to the keytab file.
- `principal` (str): The unique identity to which Kerberos can assign tickets.
- `vault_id`: (str): The OCID of the vault.

#### 16.4.1.1.1 Save

The `BDSSecretKeeper.save` API serializes and stores the credentials to Vault using the following parameters:

- `defined_tags` (dict, optional): Default None. Save the tags under predefined tags in the OCI Console.
- `description` (str) – Description of the secret when saved in Vault.
- `freeform_tags` (dict, optional): Default None. Free form tags to use for saving the secret in the OCI Console.
- `name` (str): Name of the secret when saved in Vault.
- `save_files` (bool, optional): Default True. If set to True, then the keytab and kerb5 config files are serialized and saved.

### 16.4.1.2 Examples

#### 16.4.1.2.1 With the Keytab and kerb5 Config Files

```
import ads
import fsspec
import os

from ads.secrets.big_data_service import BDSSecretKeeper
from ads.bds.auth import has_kerberos_ticket, refresh_ticket, krbcontext

ads.set_auth('resource_principal')

principal = "<your_principal>"
```

(continues on next page)

(continued from previous page)

```

hdfs_host = "<your_hdfs_host>"
hive_host = "<your_hive_host>"
hdfs_port = <your_hdfs_port>
hive_port = <your_hive_port>
vault_id = "ocid1.vault..<unique_ID>"
key_id = "ocid1.key..<unique_ID>"

secret = BDSSecretKeeper(
    vault_id=vault_id,
    key_id=key_id,
    principal=principal,
    hdfs_host=hdfs_host,
    hive_host=hive_host,
    hdfs_port=hdfs_port,
    hive_port=hive_port,
    keytab_path=keytab_path,
    kerb5_path=kerb5_path
)

saved_secret = secret.save(name="your_bds_config_secret_name",
                           description="your bds credentials",
                           freeform_tags={"schema":"emp"},
                           defined_tags={},
                           save_files=True)

```

#### 16.4.1.2.2 Without the Keytab and kerb5 Config Files

```

import ads
import fsspec
import os

from ads.secrets.big_data_service import BDSSecretKeeper
from ads.bds.auth import has_kerberos_ticket, refresh_ticket, krbcontext

ads.set_auth('resource_principal')

principal = "<your_principal>"
hdfs_host = "<your_hdfs_host>"
hive_host = "<your_hive_host>"
hdfs_port = <your_hdfs_port>
hive_port = <your_hive_port>
vault_id = "ocid1.vault..<unique_ID>"
key_id = "ocid1.key..<unique_ID>"

bds_keeper = BDSSecretKeeper(
    vault_id=vault_id,
    key_id=key_id,
    principal=principal,
    hdfs_host=hdfs_host,
    hive_host=hive_host,

```

(continues on next page)

(continued from previous page)

```

        hdfs_port=hdfs_port,
        hive_port=hive_port,
        keytab_path=keytab_path,
        kerb5_path=kerb5_path
    )

saved_secret = bds_keeper.save(name="your_bds_config_secret_name",
                               description="your bds credentials",
                               freeform_tags={"schema":"emp"},
                               defined_tags={},
                               save_files=False)

print(saved_secret.secret_id)

```

```
'ocid1.vaultsecret..<unique_ID>'
```

## 16.4.2 Load Credentials

### 16.4.2.1 Load

The `BDSecretKeeper.load_secret` API deserializes and loads the credentials from Vault. You could use this API in one of the following ways:

#### 16.4.2.1.1 Using a `with` Statement

```

with BDSecretKeeper.load_secret('ocid1.vaultsecret..<unique_ID>') as bdssecret:
    print(bdssecret['hdfs_host'])

```

This approach is preferred as the secrets are only available within the code block and it reduces the risk that the variable will be leaked.

#### 16.4.2.1.2 Without Using a `with` Statement

```

bdssecretobj = BDSecretKeeper.load_secret('ocid1.vaultsecret..<unique_ID>')
bdssecret = bdssecretobj.to_dict()
print(bdssecret['hdfs_host'])

```

The `.load_secret()` method takes following parameters:

- `auth`: Provide overriding authorization information if the authorization information is different from the `ads.set_auth` setting.
- `export_env`: Default is `False`. If set to `True`, the credentials are exported as environment variable when used with the `with` operator.
- `export_prefix`: The default name for environment variable is `user_name`, `password`, `service_name`, and `wallet_location`. You can add a prefix to avoid name collision
- `format`: Optional. If `source` is a file, then this value must be `json` or `yaml` depending on the file format.
- `keytab_dir`: Optional. Directory path where the keytab ZIP file is saved after the contents are retrieved from the vault. If the keytab content is not available in the specified secret OCID, then this attribute is ignored.



- `source`: Either the file that was exported from `export_vault_details` or the OCID of the secret

If the `keytab` and `kerb5` configuration files were saved in the vault, then a `keytab` and `kerb5` configuration file of the same name is created by `.load_secret()`. By default, the `keytab` file is created in the `keytab_path` specified in the secret. To update the location, set the directory path with `key_dir`. However, the `kerb5` configuration file is always saved in the `~/bds_config/krb5.conf` path.

Note that `keytab` and `kerb5` configuration files are saved only when the content is saved into the vault.

After you load and save the configuration parameters files, you can call the `krbcontext` context manager to create a Kerberos ticket.

## 16.4.2.2 Examples

### 16.4.2.2.1 Using a With Statement

To specify a local `keytab` file, set the path to the ZIP file with `wallet_location`:

```
from pyhive import hive

with BDSSecretKeeper.load_secret(saved_secret.secret_id, keytab_dir="~/path/to/save/
↪keytab_file/") as cred:
    with krbcontext(principal=cred["principal"], keytab_path=cred['keytab_path']):
        hive_cursor = hive.connect(host=cred["hive_host"],
                                   port=cred["hive_port"],
                                   auth='KERBEROS',
                                   kerberos_service_name="hive").cursor()
```

Now you can query the data from Hive:

```
hive_cursor.execute("""
    select *
    from your_db.your_table
    limit 10
""")

import pandas as pd
pd.DataFrame(hive_cursor.fetchall(), columns=[col[0] for col in hive_cursor.description])
```

### 16.4.2.2.2 Without Using a With Statement

#### Load From Secret OCID

```
bdssecretobj = BDSSecretKeeper.load_secret(saved_secret.secret_id)
bdssecret = bdssecretobj.to_dict()
print(bdssecret)
```

### Load From a JSON File

```
bdssecretobj = BDSSecretKeeper.load_secret(source="./my_bds_vault_info.json", format=
↪ "json")
bdssecretobj.to_dict()
```

### Load From a YAML File

```
bdssecretobj = BDSSecretKeeper.load_secret(source="./my_bds_vault_info.yaml", format=
↪ "yaml")
bdssecretobj.to_dict()
```

## 16.5 MySQL

To connect to a MySQL Database, you need the following:

- hostname
- password
- port, the default is 3306
- user name

The `MySQLDBSecretKeeper` class saves the MySQL database credentials to the OCI Vault service.

See [API Documentation](#) for more details

### 16.5.1 Save Credentials

#### 16.5.1.1 MySQLDBSecretKeeper

The `MySQLDBSecretKeeper` constructor has the following parameters:

- `compartment_id` (str): OCID of the compartment where the vault is located. Defaults to the compartment of the notebook session when used in a Data Science notebook session.
- `database` (str, optional): The database name if available.
- `host` (str): The hostname of the database.
- `key_id` (str): OCID of the master key used for encrypting the secret.
- `password` (str): The password of the database.
- `port` (str, optional). Default 3306: Port number of the database service.
- `user_name` (str): The user name to be stored.
- `vault_id` (str): OCID of the vault.

### 16.5.1.1.1 Save

The `MySQLDBSecretKeeper.save` API serializes and stores the credentials to the vault using the following parameters:

- `defined_tags` (dict, optional): Save the tags under predefined tags in the OCI Console.
- `description` (str): Description of the secret when saved in the vault.
- `freeform_tags` (dict, optional): Freeform tags to be used for saving the secret in the OCI Console.
- `name` (str): Name of the secret when saved in the vault.

The secret has the following information:

- `database`
- `host`
- `password`
- `port`
- `user_name`

### 16.5.1.2 Examples

#### 16.5.1.2.1 Save Credentials

```
import ads
from ads.secrets.mysqlldb import MySQLDBSecretKeeper

vault_id = "ocid1.vault..<unique_ID>"
key_id = "ocid1.key..<unique_ID>"

ads.set_auth("resource_principal") # If using resource principal for authentication
connection_parameters={
    "user_name": "<your user name>",
    "password": "<your password>",
    "service_name": "service_name",
    "host": "<db host>",
    "port": "<db port>",
}

mysqlldb_keeper = MySQLDBSecretKeeper(vault_id=vault_id,
                                       key_id=key_id,
                                       **connection_parameters)

mysqlldb_keeper.save("mysqlldb_employee", "My DB credentials", freeform_tags={"schema": "emp
→"})
print(mysqlldb_keeper.secret_id) # Prints the secret_id of the stored credentials
```

```
'ocid1.vaultsecret..<unique_ID>'
```

You can save the vault details in a file for later reference, or use it in your code using `export_vault_details` API calls. The API currently enables you to export the information as a YAML file or a JSON file.

```
mysqlldb_keeper.export_vault_details("my_db_vault_info.json", format="json")
```

#### 16.5.1.2.2 Save as a YAML File

```
mysqlldb_keeper.export_vault_details("my_db_vault_info.yaml", format="yaml")
```

### 16.5.2 Load Credentials

#### 16.5.2.1 Load

The `MySQLDBSecretKeeper.load_secret()` API deserializes and loads the credentials from the vault. You could use this API in one of the following ways:

##### 16.5.2.1.1 Using a `with` Statement

```
with MySQLDBSecretKeeper.load_secret('ocid1.vaultsecret..<unique_ID>') as mysqlldb_secret:
    print(mysqlldb_secret['user_name'])
```

##### 16.5.2.1.2 Without Using a `with` Statement

```
mysqlldb_secretobj = MySQLDBSecretKeeper.load_secret('ocid1.vaultsecret..<unique_ID>')
mysqlldb_secret = mysqlldb_secretobj.to_dict()
print(mysqlldb_secret['user_name'])
```

The `.load_secret()` method has the following parameters:

- `auth`: Provide overriding `auth` information if the `auth` information is different from the `ads.set_auth` setting.
- `export_env`: The default is `False`. If set to `True`, the credentials are exported as environment variable when used with the `with` operator.
- `export_prefix`: The default name for environment variable is `user_name`, `password`, `service_name`, and `wallet_location`. You can add a prefix to avoid name collision.
- `format`: (Optional) If `source` is a file, then this value must be `json` or `yaml` depending on the file format.
- `source`: Either the file that was exported from `export_vault_details`, or the OCID of the secret.

#### 16.5.2.2 Examples

##### 16.5.2.2.1 Using a `with` Statement

```
import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.mysqlldb import MySQLDBSecretKeeper

with MySQLDBSecretKeeper.load_secret(
```

(continues on next page)

(continued from previous page)

```

        "ocid1.vaultsecret..<unique_ID>"
    ) as mysqlcred2:
    print (mysqlcred2["user_name"]) # Prints the user name

print (mysqlcred2["user_name"]) # Prints nothing. The credentials are cleared from
↳the dictionary outside the ``with`` block

```

### 16.5.2.2.2 Export the Environment Variables Using a with Statement

To expose credentials as an environment variable, set `export_env=True`. The following keys are exported:

Secret attribute	Environment Variable Name
user_name	user_name
password	password
host	host
port	port
database	database

```

import os
import ads

ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.mysqlcred import MySQLDBSecretKeeper

with MySQLDBSecretKeeper.load_secret(
    "ocid1.vaultsecret..<unique_ID>",
    export_env=True
):
    print(os.environ.get("user_name")) # Prints the user name

print(os.environ.get("user_name")) # Prints nothing. The credentials are cleared from
↳the dictionary outside the ``with`` block

```

You can avoid name collisions by setting a prefix string using `export_prefix` along with `export_env=True`. For example, if you set prefix as `myprocess`, then the exported keys are:

Secret attribute	Environment Variable Name
user_name	myprocess.user_name
password	myprocess.password
host	myprocess.host
port	myprocess.port
database	myprocess.database

```

import os
import ads

ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.mysqlcred import MySQLDBSecretKeeper

```

(continues on next page)

(continued from previous page)

```
with MySQLDBSecretKeeper.load_secret(  
    "ocidl.vaultsecret.<unique_ID>",  
    export_env=True,  
    export_prefix="myprocess"  
):  
    print(os.environ.get("myprocess.user_name")) # Prints the user name  
  
print(os.environ.get("myprocess.user_name")) # Prints nothing. The credentials are_  
↪cleared from the dictionary outside the ``with`` block
```

## 16.6 Oracle Database

To connect to an Oracle Database you need the following:

- hostname
- password
- port. Default is 1521
- service name or sid
- user name

The `OracleDBSecretKeeper` class saves the Oracle Database credentials to the OCI Vault service.

See [API Documentation](#) for more details

### 16.6.1 Save Credentials

#### 16.6.1.1 OracleDBSecretKeeper

The `OracleDBSecretKeeper` constructor has the following parameters:

- `compartment_id` (str): OCID of the compartment where the vault is located. This defaults to the compartment of the notebook session when used in a Data Science notebook session.
- `dsn` (str, optional): The DSN string if available.
- `host` (str): The hostname of the database.
- `key_id` (str): OCID of the master key used for encrypting the secret.
- `password` (str): The password of the database.
- `port` (str, optional). Default 1521. Port number of the database service.
- `service_name` (str, optional): The service name of the database.
- `sid` (str, optional): The SID of the database if the service name is not available.
- `user_name` (str): The user name to be stored.
- `vault_id` (str): OCID of the vault.

### 16.6.1.2 Save

The `OracleDBSecretKeeper.save()` API serializes and stores the credentials to Vault using the following parameters:

- `defined_tags` (dict, optional): Save the tags under predefined tags in the OCI Console.
- `description` (str): Description of the secret when saved in the vault.
- `freeform_tags` (dict, optional): Freeform tags to use when saving the secret in the OCI Console.
- `name` (str): Name of the secret when saved in the vault.

The secret has the following information:

- `dsn`
- `host`
- `password`
- `port`
- `service_name`
- `sid`
- `user_name`

### 16.6.1.3 Examples

#### 16.6.1.3.1 Save Credentials

```
import ads
from ads.secrets.oracledb import OracleDBSecretKeeper

vault_id = "ocid1.vault.<unique_ID>"
key_id = "ocid1.key.<unique_ID>"

ads.set_auth("resource_principal") # If using resource principal for authentication
connection_parameters={
    "user_name": "<your user name>",
    "password": "<your password>",
    "service_name": "service_name",
    "host": "<db host>",
    "port": "<db port>",
}

oracledb_keeper = OracleDBSecretKeeper(vault_id=vault_id,
                                       key_id=key_id,
                                       **connection_parameters)

oracledb_keeper.save("oracledb_employee", "My DB credentials", freeform_tags={"schema":
↪ "emp"})
print(oracledb_keeper.secret_id) # Prints the secret_id of the stored credentials

'ocid1.vaultsecret.<unique_ID>'
```

You can save the vault details in a file for later reference or using it within your code using `export_vault_details` API calls. The API currently enables you to export the information as a YAML file or a JSON file.

```
oracledb_keeper.export_vault_details("my_db_vault_info.json", format="json")
```

#### 16.6.1.3.2 Save as a YAML File

```
oracledb_keeper.export_vault_details("my_db_vault_info.yaml", format="yaml")
```

### 16.6.2 Load Credentials

#### 16.6.2.1 Load

The `OracleDBSecretKeeper.load_secret()` API deserializes and loads the credentials from the vault. You could use this API in one of the following ways:

##### 16.6.2.1.1 Using a with Statement

```
with OracleDBSecretKeeper.load_secret('ocidl.vaultsecret.<unique_ID>') as oracledb_
↪secret:
    print(oracledb_secret['user_name'])
```

##### 16.6.2.1.2 Without using a with Statement

```
oracledb_secretobj = OracleDBSecretKeeper.load_secret('ocidl.vaultsecret.<unique_ID>')
oracledb_secret = oracledb_secretobj.to_dict()
print(oracledb_secret['user_name'])
```

The `.load_secret()` method has the following parameters:

- **auth:** Provide overriding authorization information if the authorization information is different from the `ads.set_auth` setting.
- **export\_env:** Default is `False`. If set to `True`, the credentials are exported as environment variable when used with the `with` operator.
- **export\_prefix:** The default name for environment variable is `user_name`, `password`, `service_name`, and `wallet_location`. You can add a prefix to avoid name collision.
- **format:** Optional. If `source` is a file, then this value must be `json` or `yaml` depending on the file format.
- **source:** Either the file that was exported from `export_vault_details` or the OCID of the secret



## 16.6.2.2 Examples

### 16.6.2.2.1 Using a with Statement

```
import ads
ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.oracledb import OracleDBSecretKeeper

with OracleDBSecretKeeper.load_secret(
    "ocid1.vaultsecret.<unique_ID>"
) as oracledb_creds2:
    print (oracledb_creds2["user_name"]) # Prints the user name

print (oracledb_creds2["user_name"]) # Prints nothing. The credentials are cleared from
↳ the dictionary outside the ``with`` block
```

### 16.6.2.2.2 Export the Environment Variable Using a with Statement

To expose credentials as an environment variable, set `export_env=True`. The following keys are exported:

Secret attribute	Environment Variable Name
user_name	user_name
password	password
host	host
port	port
service user_name	service_name
sid	sid
dsn	dsn

```
import os
import ads

ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.oracledb import OracleDBSecretKeeper

with OracleDBSecretKeeper.load_secret(
    "ocid1.vaultsecret.<unique_ID>",
    export_env=True
):
    print(os.environ.get("user_name")) # Prints the user name

print(os.environ.get("user_name")) # Prints nothing. The credentials are cleared from
↳ the dictionary outside the ``with`` block
```

You can avoid name collisions by setting a prefix string using `export_prefix` along with `export_env=True`. For example, if you set prefix as `myprocess`, then the exported keys are:

Secret attribute	Environment Variable Name
user_name	myprocess.user_name
password	myprocess.password
host	myprocess.host
port	myprocess.port
service user_name	myprocess.service_name
sid	myprocess.sid
dsn	myprocess.dsn

```
import os
import ads

ads.set_auth('resource_principal') # If using resource principal authentication
from ads.secrets.oracledb import OracleDBSecretKeeper

with OracleDBSecretKeeper.load_secret(
    "ocid1.vaultsecret..<unique_ID>",
    export_env=True,
    export_prefix="myprocess"
):
    print(os.environ.get("myprocess.user_name")) # Prints the user name

print(os.environ.get("myprocess.user_name")) # Prints nothing. The credentials are
↪cleared from the dictionary outside the ``with`` block
```

## MODEL VERSION SET

### 17.1 Overview

The normal workflow of a data scientist is to create a model and push it into production. While in production the data scientist learns what the model is doing well and what it isn't. Using this information the data scientist creates an improved model. These models are linked using model version sets. A model version set is a collection of models that are related to each other. A model version set is a way to track the relationships between models. As a container, the model version set takes a collection of models. Those models are assigned a sequential version number based on the order they are entered into the model version set.

In ADS the class `ModelVersionSet` is used to represent the model version set. An object of `ModelVersionSet` references a model version set in the Data Science service. The `ModelVersionSet` class supports two APIs: the builder pattern and the traditional parameter-based pattern. You can use either of these API frameworks interchangeably and examples for both patterns are included.

Use the `.create()` method to create a model version set in your tenancy. If the model version set already exists in the model catalog, use the `.from_id()` or `.from_name()` method to get a `ModelVersionSet` object based on the specified model version set. If you make changes to the metadata associated with the model version set, use the `.update()` method to push those changes to the model catalog. The `.list()` method lists all model version sets. To add an existing model to a model version set, use the `.add_model()` method. The `.models()` method lists the models in the model version set. Use the `.delete()` method to delete a model version set from the model catalog.

### 17.2 Quick Start

The following creates a model and model version set, and then performs some common operations on the model version set:

```
import tempfile
from ads.model import SklearnModel
from ads.model import ModelVersionSet
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Create a model version set
mvs = ModelVersionSet(
    name = "my_test_model_version_set",
    description = "A test creating the model version set using ModelVersionSet")
mvs.create()
```

(continues on next page)

(continued from previous page)

```

# Create a Sklearn model
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
sklearn_estimator = LogisticRegression()
sklearn_estimator.fit(X_train, y_train)

# Create an SklearnModel object
sklearn_model = SklearnModel(estimator=sklearn_estimator, artifact_dir=tempfile.
    ↪mkdtemp())
sklearn_model.prepare(inference_conda_env="dbexp_p38_cpu_v1")

# Save the model and add it to the model version set
model_id = sklearn_model.save(
    display_name="Quickstart model",
    model_version_set=mvs,
    version_label="Version 1")

# Print a list of models in the model version set
for item in ModelVersionSet.list():
    print(item)
    print("-----")

# Update the model version set
mvs.description = "Updated description of the model version set"
mvs.update()

# Delete the model version set and associated models
# mvs.delete(delete_model=True)

```

## 17.3 Associate a Model

Model version sets are a collection of models. After a model is associated with a model version set, the model can't be associated with a different model version set. Further, the model can't be disassociated with the model version set.

When a model is associated with a model version set, a version label can be assigned to the set. This version is different than the model version that is maintained by the model version set.

There are a number of ways to associate a model with a model version set. Which approach you use depends on the workflow.

### 17.3.1 ModelVersionSet Object

For a model not associated with a model version set, use the `.model_add()` method on a `ModelVersionSet` object to associate the model with the model version set. The `.model_add()` requires that you provide the model OCID and optionally a version label.

```
mvs = ModelVersionSet.from_id(id="<model_version_set_id>")
mvs.model_add(<your_model_id>, version_label="Version 1")
```

### 17.3.2 Model Serialization

The Model Serialization classes allow a model to be associated with a model version set at the time that it is saved to the model catalog. You do this with the `model_version_set` parameter in the `.save()` method. In addition, you can add the model's version label with the `version_label` parameter.

The `model_version_set` parameter accepts a model version set's OCID or name. The parameter also accepts a `ModelVersionSet` object.

In the following, the `model` variable is a Model Serialization object that is to be saved to the model catalog, and at the same time associated with a model version set.

```
model.save(
    display_name='Model attached to a model version set',
    version_label = "Version 1",
    model_version_set="<model_version_set_id>"
)
```

### 17.3.3 Context Manager

To associate several models with a model version set, use a context manager. The `ads.model.experiment()` method requires a `name` parameter. If the model catalog has a matching model version set name, the model catalog uses that model version set. If the parameter `create_if_not_exists` is `True`, the `experiment()` method attempts to match the model version set name with `name` in the model catalog. If the name does not exist, the method creates a new model version set.

Within the context manager, you can save multiple Model Serialization models without specifying the `model_version_set` parameter because it's taken from the model context manager. The following example assumes that `model_1`, `model_2`, and `model_3` are Model Serialization objects. If the model version set doesn't exist in the model catalog, the example creates a model version set named `my_model_version_set`. If the model version set exists in the model catalog, the models are saved to that model version set.

```
with ads.model.experiment(name="my_model_version_set", create_if_not_exists=True):
    # experiment 1
    model_1.save(
        display_name='Generic Model Experiment 1',
        version_label = "Experiment 1"
    )

    # experiment 2
    model_2.save(
        display_name='Generic Model Experiment 2',
        version_label = "Experiment 2"
```

(continues on next page)

(continued from previous page)

```
)

# experiment 3
model_3.save(
    display_name='Generic Model Experiment 3',
    version_label = "Experiment 3"
)
```

## 17.4 Create

The `.create()` method on a `ModelVersionSet` object creates a model version set in the model catalog. The properties of the `ModelVersionSet` are used to create the model version set in the model catalog.

The following examples create a `ModelVersionSet`, define the properties of the model version set, and then create a model version set in the model catalog.

### 17.4.1 Parameter-based Pattern

```
mvs = ModelVersionSet(
    compartment_id = os.environ["PROJECT_COMPARTMENT_OCID"],
    name = "my_model_version_set",
    projectId = os.environ["PROJECT_OCID"],
    description = "Sample model version set")
mvs.create()
```

### 17.4.2 Builder Pattern

```
mvs = (ModelVersionSet()
      .with_compartment_id(os.environ["PROJECT_COMPARTMENT_OCID"])
      .with_project_id(os.environ["PROJECT_OCID"])
      .with_name("my_model_version_set")
      .with_description("Sample model version set"))
mvs.create()
```

## 17.5 Delete

To delete a model version set, all the associated models must be deleted or in a terminated state. You can set the `delete_model` parameter to `True` to delete all of the models in the model version set, and then delete the model version set. The `.delete()` method on a `ModelVersionSet` object initiates an asynchronous delete operation. You can check the `.status` method on the `ModelVersionSet` object to determine the status of the delete request.

The following example deletes a model version set and its associated models.

The status property has the following values:

- `ModelVersionSet.LIFECYCLE_STATE_ACTIVE`
- `ModelVersionSet.LIFECYCLE_STATE_DELETED`

- `ModelVersionSet.LIFECYCLE_STATE_DELETING`
- `ModelVersionSet.LIFECYCLE_STATE_FAILED`

## 17.6 Download

Create a `ModelVersionSet` object by downloading the metadata from the model catalog. The `ModelVersionSet` class has a `.from_id()` method that accepts the model version set OCID. The `.from_name()` method takes the name of the model version set.

### 17.6.1 `.from_id()`

```
mvs = ModelVersionSet.from_id(id="<model_version_set_id>")
```

### 17.6.2 `.from_name()`

```
mvs = ModelVersionSet.from_name(name="<model_version_set_name>")
```

## 17.7 List

### 17.7.1 `ModelVersionSet`

The `.list()` method on the `ModelVersionSet` class takes a compartment ID and lists the model version sets in that compartment. If the compartment isn't given, then the compartment of the notebook session is used.

The following example uses context manager to iterate over the collection of model version sets:

```
for model_version_set in ModelVersionSet.list():
    print(model_version_set)
    print("-----")
```

### 17.7.2 `Model`

You can get the list of models associated with a model version set by calling the `.models()` method on a `ModelVersionSet` object. A list of models that are associated with that model version set is returned. First, you must obtain a `ModelVersionSet` object. Use the `.from_id()` method if you know the model version set OCID. Alternatively, use the `.from_name()` method if you know the name of the model version set.

```
mvs = ModelVersionSet.from_id(id="<model_version_set_id>")
models = mvs.models()

for dsc_model in models:
    print(dsc_model.display_name, dsc_model.id, dsc_model.status)
```

## 17.8 Update

### 17.8.1 ModelVersionSet Properties

The `ModelVersionSet` object has a number of properties that you can be update. When the properties in a `ModelVersionSet` object are updated, the model version set in the model catalog are not automatically updated. You must call the `.update()` method to commit the changes.

The properties that you can be update are:

- `compartment_id`: The OCID of the compartment that the model version set belongs to.
- `description`: A description of the models in the collection.
- `freeform_tags`: A dictionary of string values.
- `name`: Name of the model version set.
- `project_id`: The OCID of the data science project that the model version set belongs to.

The following demonstrates how to update these values of a model version set using the various API interfaces:

#### 17.8.1.1 Parameter-based Pattern

```
mvs = ModelVersionSet.from_id(id="<model_version_set_id>")
mvs.compartement_id = os.environ["PROJECT_COMPARTMENT_OCID"]
mvs.description = "An updated description"
mvs.freeform_tags = {'label_1': 'value 1', 'label_2': 'value 2'}
mvs.name = "new_set_name"
mvs.project_id = os.environ["PROJECT_OCID"]
mvs.update()
```

#### 17.8.1.2 Builder Pattern

```
mvs = ModelVersionSet.from_id(id="<model_version_set_id>")
mvs = (mvs.with_compartment_id(os.environ["PROJECT_COMPARTMENT_OCID"]))
    .with_description("An updated description")
    .with_freeform_tags(label_1="value 1", label_2="value 2")
    .with_name("new_set_name")
    .with_project_id(os.environ["PROJECT_OCID"])
    .update()
```

### 17.8.2 Version Label

The version label is associated with the model, and not the model version set. To change the version label, you must have a `Model` object. Then, you can change the `version_label` for the registered model.

The following example gets a registered `Model` object by model's OCID. Then, the object updates the version label property.



```
from ads.model import LightGBMModel

lgbm_model = LightGBMModel.from_id(
    "ocid1.datasciencemodel.oc1.xxx.xxxxx",
    model_file_name="model.joblib",
    artifact_dir="lgbm-download-test",
    force_overwrite=True,
)

lgbm_model.update(version_label="MyNewVersionLabel")
```



## CLASS DOCUMENTATION

### 18.1 ads package

#### 18.1.1 Subpackages

##### 18.1.1.1 ads.automl package

###### 18.1.1.1.1 Submodules

###### 18.1.1.1.2 ads.automl.driver module

**class** `ads.automl.driver.AutoML`(*training\_data*, *validation\_data=None*, *provider=None*, *baseline='dummy'*, *client=None*)

Bases: `object`

Creates an Automatic machine learning object.

#### Parameters

- **training\_data** (*ADSDData* instance) –
- **validation\_data** (*ADSDData* instance) –
- **provider** (*None* or object of `ads.automl.provider.AutoMLProvider`) – If *None*, the default `OracleAutoMLProvider` will be used to generate the model
- **baseline** (*None*, *"dummy"*, or object of `ads.common.model.ADSModel` (Default is *"dummy"*)) –
  - If *None*, than no baseline is created,
  - If *"dummy"*, than the `DummyClassifier` or `DummyRegressor` are used
  - If Object, than whatever estimator is provided will be used.

This estimator must include a part of its pipeline which does preprocessing to handle categorical data

- **client** – Dask Client to use (optional)

## Examples

```
>>> train, test = ds.train_test_split()
>>> olabs_automl = OracleAutoMLProvider()
>>> model, baseline = AutoML(train, provider=olabs_automl).train()
```

**train(\*\*kwargs)**

Returns a fitted automl model and a fitted baseline model.

### Parameters

**kwargs** (*dict*, *optional*) – kwargs passed to provider’s train method

### Returns

- **model** (*object of ads.common.model.ADSModel*) – the trained automl model
- **baseline** (*object of ads.common.model.ADSModel*) – the baseline model to compare

## Examples

```
>>> train, test = ds.train_test_split()
>>> olabs_automl = OracleAutoMLProvider()
>>> model, baseline = AutoML(train, provider=olabs_automl).train()
```

`ads.automl.driver.get_ml_task_type(X, y, classes)`

Gets the ML task type and returns it.

### Parameters

- **X** (*Dataframe*) – The training dataframe
- **Y** (*Dataframe*) – The testing dataframe
- **Classes** (*List*) – a list of classes

### Returns

A particular task type like *REGRESSION*, *MULTI\_CLASS\_CLASSIFICATION*...

### Return type

ml\_task\_type

### 18.1.1.1.3 ads.automl.provider module

**class** `ads.automl.provider.AutoMLFeatureSelection(msg)`

Bases: `object`

**fit**(X)

Fits the baseline estimator

### Parameters

**X** (*Dataframe or list-like*) – A Dataframe or list-like object holding data to be predicted on

### Returns

**Self** – The fitted estimator

### Return type

Estimator

**transform(X)**

Runs the Baselines transform function and returns the result

**Parameters**

**X** (*Dataframe or list-like*) – A Dataframe or list-like object holding data to be transformed

**Returns**

**X** – The transformed Dataframe.

**Return type**

Dataframe or list-like

**class** ads.automl.provider.**AutoMLPreprocessingTransformer**(msg)

Bases: object

**fit(X)**

Fits the preprocessing Transformer

**Parameters**

**X** (*Dataframe or list-like*) – A Dataframe or list-like object holding data to be predicted on

**Returns**

**Self** – The fitted estimator

**Return type**

Estimator

**transform(X)**

Runs the preprocessing transform function and returns the result

**Parameters**

**X** (*Dataframe or list-like*) – A Dataframe or list-like object holding data to be transformed

**Returns**

**X** – The transformed Dataframe.

**Return type**

Dataframe or list-like

**class** ads.automl.provider.**AutoMLProvider**

Bases: ABC

Abstract Base Class defining the structure of an AutoML solution. The solution needs to implement train() and get\_transformer\_pipeline().

**property est**

Returns the estimator.

The estimator can be a standard sklearn estimator or any object that implement methods from (BaseEstimator, RegressorMixin) for regression or (BaseEstimator, ClassifierMixin) for classification.

**Returns**

**est**

**Return type**

An instance of estimator

**abstract get\_transformer\_pipeline()**

Returns a list of transformers representing the transformations done on data before model prediction.

This method is optional to implement, and is used only for visualizing transformations on data using `ADSModel#visualize_transforms()`.

**Returns**

**transformers\_list**

**Return type**

list of transformers implementing fit and transform

**setup**(*X\_train*, *y\_train*, *ml\_task\_type*, *X\_valid=None*, *y\_valid=None*, *class\_names=None*, *client=None*)

Setup arguments to the AutoML instance.

**Parameters**

- **X\_train** (*DataFrame*) – Training features
- **y\_train** (*DataFrame*) – Training labels
- **ml\_task\_type** (*One of ml\_task\_type.{REGRESSION,BINARY\_CLASSIFICATION,} –MULTI\_CLASS\_CLASSIFICATION,BINARY\_TEXT\_CLASSIFICATION,MULTI\_CLASS\_TEXT\_CLASS*)
- **X\_valid** (*DataFrame*) – Validation features
- **y\_valid** (*DataFrame*) – Validation labels
- **class\_names** (*list*) – Unique values in *y\_train*
- **client** (*object*) – Dask client instance for distributed execution

**abstract train**(*\*\*kwargs*)

Calls fit on estimator.

This method is expected to set the ‘est’ property.

**Parameters**

- **kwargs** (*dict, optional*) –
- **method** (*kwargs to decide the estimator and arguments for the fit*) –

**class** `ads.automl.provider.BaselineAutoMLProvider`(*est*)

Bases: [\*AutoMLProvider\*](#)

Generates a baseline model using the Zero Rule algorithm by default. For a classification predictive modeling problem where a categorical value is predicted, the Zero Rule algorithm predicts the class value that has the most observations in the training dataset.

**Parameters**

**est** ([\*BaselineModel\*](#)) – An estimator that supports the fit/predict/predict\_proba interface. By default, `DummyClassifier/DummyRegressor` are used as estimators

**decide\_estimator**(*\*\*kwargs*)

Decides which type of `BaselineModel` to generate.

**Returns**

**Modell** – A baseline model generated for the particular ML task being performed

**Return type**

[\*BaselineModel\*](#)

**get\_transformer\_pipeline()**

Returns a list of transformers representing the transformations done on data before model prediction.

This method is used only for visualizing transformations on data using `ADSModel#visualize_transforms()`.

**Returns**

**transformers\_list**

**Return type**

list of transformers implementing fit and transform

**train(\*\*kwargs)**

Calls fit on estimator.

This method is expected to set the 'est' property.

**Parameters**

- **kwargs** (*dict, optional*) –
- **method** (*kwargs to decide the estimator and arguments for the fit*) –

**class ads.automl.provider.BaselineModel(est)**

Bases: object

A BaselineModel object that supports fit/predict/predict\_proba/transform interface. Labels (y) are encoded using DataFrameLabelEncoder.

**fit(X, y)**

Fits the baseline estimator.

**Parameters**

- **X** (*Dataframe or list-like*) – A Dataframe or list-like object holding data to be predicted on
- **Y** (*Dataframe, Series, or list-like*) – A Dataframe, series, or list-like object holding the labels

**Returns**

**estimator**

**Return type**

The fitted estimator

**predict(X)**

Runs the Baselines predict function and returns the result.

**Parameters**

**X** (*Dataframe or list-like*) – A Dataframe or list-like object holding data to be predicted on

**Returns**

**List**

**Return type**

A list of predictions performed on the input data.

**predict\_proba(X)**

Runs the Baselines predict\_proba function and returns the result.

**Parameters**

**X** (*Dataframe or list-like*) – A Dataframe or list-like object holding data to be predicted on

**Returns****List****Return type**

A list of probabilities of being part of a class

**transform(X)**

Runs the Baselines transform function and returns the result.

**Parameters****X** (*Dataframe or list-like*) – A Dataframe or list-like object holding data to be transformed**Returns****Dataframe or list-like****Return type**

The transformed Dataframe. Currently, no transformation is performed by the default Baseline Estimator.

```
class ads.automl.provider.OracleAutoMLProvider(n_jobs=-1, loglevel=None, logger_override=None,  
                                              model_n_jobs: int = 1)
```

Bases: [AutoMLProvider](#), ABC

The Oracle AutoML Provider automatically provides a tuned ML pipeline that best models the given a training dataset and a prediction task at hand.

**Parameters**

- **n\_jobs** (*int*) – Specifies the degree of parallelism for Oracle AutoML. -1 (default) means that AutoML will use all available cores.
- **loglevel** (*int*) – The verbosity of output for Oracle AutoML. Can be specified using the Python logging module (<https://docs.python.org/3/library/logging.html#logging-levels>).
- **model\_n\_jobs** (*(optional, int)*). Defaults to 1.) – Specifies the model parallelism used by AutoML. This will be passed to the underlying model it is training.

**get\_transformer\_pipeline()**

Returns a list of transformers representing the transformations done on data before model prediction.

This method is used only for visualizing transformations on data using `ADSModel#visualize_transforms()`.**Returns****transformers\_list****Return type**

list of transformers implementing fit and transform

```
print_summary(max_rows=None, sort_column='Mean Validation Score', ranking_table_only=False)
```

Prints a summary of the Oracle AutoML Pipeline in the last `train()` call.**Parameters**

- **max\_rows** (*int*) – Number of trials to print. Pass in None to print all trials
- **sort\_column** (*string*) – Column to sort results by. Must be one of ['Algorithm', '#Samples', '#Features', 'Mean Validation Score', 'Hyperparameters', 'All Validation Scores', 'CPU Time']
- **ranking\_table\_only** (*bool*) – Table to be displayed. Pass in False to display the complete table. Pass in True to display the ranking table only.



**print\_trials**(*max\_rows=None, sort\_column='Mean Validation Score'*)

Prints all trials executed by the Oracle AutoML Pipeline in the last train() call.

#### Parameters

- **max\_rows** (*int*) – Number of trials to print. Pass in None to print all trials
- **sort\_column** (*string*) – Column to sort results by. Must be one of ['Algorithm', '#Samples', '#Features', 'Mean Validation Score', 'Hyperparameters', 'All Validation Scores', 'CPU Time']

**selected\_model\_name**()

Return the name of the selected model by AutoML.

**selected\_score\_label**()

Return the name of score\_metric used in train.

**train**(\*\**kwargs*)

Train the Oracle AutoML Pipeline. This looks at the training data, and identifies the best set of features, the best algorithm and the best set of hyperparameters for this data. A model is then generated, trained on this data and returned.

#### Parameters

- **score\_metric** (*str, callable*) – Score function (or loss function) with signature `score_func(y, y_pred, **kwargs)` or string specified as [https://scikit-learn.org/stable/modules/model\\_evaluation.html#common-cases-predefined-values](https://scikit-learn.org/stable/modules/model_evaluation.html#common-cases-predefined-values)
- **random\_state** (*int*) – Random seed used by AutoML
- **model\_list** (*list of str*) – Models that will be evaluated by the Pipeline. Supported models: - Classification: AdaBoostClassifier, DecisionTreeClassifier, ExtraTreesClassifier, KNeighborsClassifier, LGBMClassifier, LinearSVC, LogisticRegression, RandomForestClassifier, SVC, XGBClassifier - Regression: AdaBoostRegressor, DecisionTreeRegressor, ExtraTreesRegressor, KNeighborsRegressor, LGBMRegressor, LinearSVR, LinearRegression, RandomForestRegressor, SVR, XGBRegressor
- **time\_budget** (*float, optional*) – Time budget in seconds where 0 means no time budget constraint (best effort)
- **min\_features** (*int, float, list, optional (default: 1)*) – Minimum number of features to keep. Acceptable values: - If int,  $0 < \text{min\_features} \leq \text{n\_features}$  - If float,  $0 < \text{min\_features} \leq 1.0$  - If list, names of features to keep, for example ['a', 'b'] means keep features 'a' and 'b'

#### Returns

self

#### Return type

object

**visualize\_adaptive\_sampling\_trials**()

Visualize the trials for Adaptive Sampling.

**visualize\_algorithm\_selection\_trials**(*ylabel=None*)

Plot the scores predicted by Algorithm Selection for each algorithm. The horizontal line shows the average score across all algorithms. Algorithms below the line are colored turquoise, whereas those with a score higher than the mean are colored teal. The orange bar shows the algorithm with the highest predicted score. The error bar is +/- one standard error.

**Parameters**

**ylabel** (*str*,) – Label for the y-axis. Defaults to the scoring metric.

**visualize\_feature\_selection\_trials**(*ylabel=None*)

Visualize the feature selection trials taken to arrive at optimal set of features. The orange line shows the optimal number of features chosen by Feature Selection.

**Parameters**

**ylabel** (*str*,) – Label for the y-axis. Defaults to the scoring metric.

**visualize\_tuning\_trials**(*ylabel=None*)

Visualize (plot) the hyperparamter tuning trials taken to arrive at the optimal hyper parameters. Each trial in the plot represents a particular hyperparamter combination.

**Parameters**

**ylabel** (*str*,) – Label for the y-axis. Defaults to the scoring metric.

**18.1.1.1.4 Module contents****18.1.1.2 ads.catalog package****18.1.1.2.1 Submodules****18.1.1.2.2 ads.catalog.model module**

**class** `ads.catalog.model.Model`(*model: Model, model\_etag: str, provenance\_metadata: ModelProvenance, provenance\_etag: str, ds\_client: DataScienceClient, identity\_client: IdentityClient*)

Bases: `object`

Class that represents the ADS implementation of model catalog item. Converts the metadata and schema from OCI implementation to ADS implementation.

**to\_dataframe()**

Converts model to dataframe format.

**show\_in\_notebook()**

Shows model in the notebook in dataframe or YAML representation.

**activate()**

Activates model.

**deactivate()**

Deactivates model.

**commit()**

Commits the changes made to the model.

**rollback()**

Rollbacks the changes made to the model.

**load\_model()**

Loads the model from the model catalog based on model ID.

Initializes the Model.

**Parameters**

- **model** (*OCIModel*) – The OCI model object.
- **model\_etag** (*str*) – The model ETag.
- **provenance\_metadata** (*ModelProvenance*) – The model provenance metadata.
- **provenance\_etag** (*str*) – The model provenance metadata ETag.
- **ds\_client** (*DataScienceClient*) – The Oracle DataScience client.
- **identity\_client** (*IdentityClient*) – The Oracle Identity Service Client.

**activate()** → None

Activates model.

**Returns**

Nothing.

**Return type**

None

**commit**(*force: bool = True*) → None

Commits model changes.

**Parameters**

**force** (*bool*) – If True, any remote changes on this model would be lost.

**Returns**

Nothing.

**Return type**

None

**deactivate()** → None

Deactivates model.

**Returns**

Nothing.

**Return type**

None

**classmethod load\_model**(*ds\_client: DataScienceClient, identity\_client: IdentityClient, model\_id: str*) → *Model*

Loads the model from the model catalog based on model ID.

**Parameters**

- **ds\_client** (*DataScienceClient*) – The Oracle DataScience client.
- **identity\_client** (*IdentityClient*) – The Oracle Identity Service Client.
- **model\_id** (*str*) – The model ID.

**Returns**

The ADS model catalog item.

**Return type**

*Model*

**Raises**

- **ServiceError** – If error occurs while getting model from server.:
- **KeyError** – If model not found.:

- **ValueError** – If error occurs while getting model provenance metatdata from server.:

**rollback()** → None

Rollbacks the changes made to the model.

**Returns**

Nothing.

**Return type**

None

**show\_in\_notebook**(*display\_format: str = 'dataframe'*) → None

Shows model in dataframe or yaml format. Supported formats: *dataframe* and *yaml*. Defaults to dataframe format.

**Returns**

Nothing.

**Return type**

None

**to\_dataframe()** → DataFrame

Converts the model to dataframe format.

**Returns**

Pandas dataframe.

**Return type**

panadas.DataFrame

**exception** `ads.catalog.model.ModelArtifactSizeError`(*max\_artifact\_size: str*)

Bases: Exception

**class** `ads.catalog.model.ModelCatalog`(*compartment\_id: Optional[str] = None, ds\_client\_auth: Optional[dict] = None, identity\_client\_auth: Optional[dict] = None, timeout: Optional[int] = None, ds\_client: Optional[DataScienceClient] = None, identity\_client: Optional[IdentityClient] = None*)

Bases: object

Allows to list, load, update, download, upload and delete models from model catalog.

**get\_model**(*self, model\_id*)

Loads the model from the model catalog based on model\_id.

**list\_models**(*self, project\_id=None, include\_deleted=False, datetime\_format=utils.date\_format, \\*\*kwargs*)

Lists all models in a given compartment, or in the current project if project\_id is specified.

**list\_model\_deployment**(*self, model\_id, config=None, tenant\_id=None, limit=500, page=None, \\*\*kwargs*)

Gets the list of model deployments by model Id across the compartments.

**update\_model**(*self, model\_id, update\_model\_details=None, \\*\*kwargs*)

Updates a model with given model\_id, using the provided update data.

**delete\_model**(*self, model, \\*\*kwargs*)

Deletes the model based on model\_id.

**download\_model**(*self*, *model\_id*, *target\_dir*, *force\_overwrite=False*, *install\_libs=False*, *conflict\_strategy=ConflictStrategy.IGNORE*)

Downloads the model from *model\_dir* to *target\_dir* based on *model\_id*.

**upload\_model**(*self*, *model\_artifact*, *provenance\_metadata=None*, *project\_id=None*, *display\_name=None*, *description=None*)

Uploads the model artifact to cloud storage.

Initializes model catalog instance.

#### Parameters

- **compartment\_id** ((*str*, *optional*). Defaults to *None*.) – Model compartment OCID. If *None*, the *config.NB\_SESSION\_COMPARTMENT\_OCID* would be used.
- **ds\_client\_auth** ((*dict*, *optional*). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate *DataServiceClient* object.
- **identity\_client\_auth** ((*dict*, *optional*). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate *IdentityClient* object.
- **timeout** ((*int*, *optional*). Defaults to 10 seconds.) – The connection timeout in seconds for the client.
- **ds\_client** (*DataServiceClient*) – The Oracle DataScience client.
- **identity\_client** (*IdentityClient*) – The Oracle Identity Service Client.

#### Raises

- **ValueError** – If compartment ID not specified.
- **TypeError** – If timeout not an integer.

**delete\_model**(*model*: *Union[str, ads.catalog.Model]*, *\*\*kwargs*) → *bool*

Deletes the model from Model Catalog.

#### Parameters

- **model** (*Union[str, "ads.catalog.Model"]*) – The OCID of the model to delete as a string, or a *ads.catalog.Model* instance.
- **kwargs** –
  - delete\_associated\_model\_deployment**: (*bool*, *optional*). Defaults to *False*. Whether associated model deployments need to be deleted or not.

#### Returns

*True* if the model was successfully deleted.

#### Return type

*bool*

#### Raises

**ModelWithActiveDeploymentError** – If model has active model deployments and inout attribute *delete\_associated\_model\_deployment* set to *False*.

**download\_model**(*model\_id*: str, *target\_dir*: str, *force\_overwrite*: bool = False, *install\_libs*: bool = False, *conflict\_strategy*=*'IGNORE'*, *bucket\_uri*: Optional[str] = None, *remove\_existing\_artifact*: Optional[bool] = True)

Downloads the model from model\_dir to target\_dir based on model\_id.

#### Parameters

- **model\_id** (str) – The OCID of the model to download.
- **target\_dir** (str) – The target location of model after download.
- **force\_overwrite** (bool) – Overwrite target\_dir if exists.
- **install\_libs** (bool, default: False) – Install the libraries specified in ds-requirements.txt which are missing in the current environment.
- **conflict\_strategy** (ConflictStrategy, default: IGNORE) – Determines how to handle version conflicts between the current environment and requirements of model artifact. Valid values: “IGNORE”, “UPDATE” or ConflictStrategy. IGNORE: Use the installed version in case of conflict UPDATE: Force update dependency to the version required by model artifact in case of conflict
- **bucket\_uri** ((str, optional). Defaults to None.) – The OCI Object Storage URI where model artifacts will be copied to. The *bucket\_uri* is only necessary for downloading large artifacts with size is greater than 2GB. Example: *oci://<bucket\_name>@<namespace>/prefix/*.
- **remove\_existing\_artifact** ((bool, optional). Defaults to True.) – Whether artifacts uploaded to object storage bucket need to be removed or not.

#### Returns

A ModelArtifact instance.

#### Return type

*ModelArtifact*

**get\_model**(*model\_id*)

Loads the model from the model catalog based on model\_id.

#### Parameters

**model\_id** (str, required) – The model ID.

#### Returns

The ads.catalog.Model with the matching ID.

#### Return type

ads.catalog.Model

**list\_model\_deployment**(*model\_id*: str, *config*: Optional[dict] = None, *tenant\_id*: Optional[str] = None, *limit*: int = 500, *page*: Optional[str] = None, *\*\*kwargs*)

Gets the list of model deployments by model Id across the compartments.

#### Parameters

- **model\_id** (str) – The model ID.
- **config** (dict (optional)) – Configuration keys and values as per SDK and Tool Configuration. The from\_file() method can be used to load configuration from a file. Alternatively, a dict can be passed. You can validate\_config the dict using validate\_config(). Defaults to None.

- **tenant\_id** (*str* *optional*) – The tenancy ID, which can be used to specify a different tenancy (for cross-tenancy authorization) when searching for resources in a different tenancy. Defaults to None.
- **limit** (*int* *optional*) – The maximum number of items to return. The value must be between 1 and 1000. Defaults to 500.
- **page** (*str* *optional*) – The page at which to start retrieving results.

**Return type**

The list of model deployments.

**list\_models**(*project\_id: Optional[str] = None, include\_deleted: bool = False, datetime\_format: str = '%Y-%m-%d %H:%M:%S', \*\*kwargs*)

Lists all models in a given compartment, or in the current project if *project\_id* is specified.

**Parameters**

- **project\_id** (*str*) – The *project\_id* of model.
- **include\_deleted** (*bool*, *optional*, *default=False*) – Whether to include deleted models in the returned list.
- **datetime\_format** (*str*, *optional*, *default: '%Y-%m-%d %H:%M:%S'*) – Change format for date time fields.

**Returns**

A list of models.

**Return type**

*ModelSummaryList*

**update\_model**(*model\_id, update\_model\_details=None, \*\*kwargs*) → *Model*

Updates a model with given *model\_id*, using the provided update data.

**Parameters**

- **model\_id** (*str*) – The model ID.
- **update\_model\_details** (*UpdateModelDetails*) – Contains the update model details data to apply. Mandatory unless *kwargs* are supplied.
- **kwargs** (*dict*, *optional*) – Update model details can be supplied instead as *kwargs*.

**Returns**

The *ads.catalog.Model* with the matching ID.

**Return type**

*Model*

**upload\_model**(*model\_artifact: ModelArtifact, provenance\_metadata: Optional[ModelProvenance] = None, project\_id: Optional[str] = None, display\_name: Optional[str] = None, description: Optional[str] = None, freeform\_tags: Optional[Dict[str, Dict[str, object]]] = None, defined\_tags: Optional[Dict[str, Dict[str, object]]] = None, bucket\_uri: Optional[str] = None, remove\_existing\_artifact: Optional[bool] = True, overwrite\_existing\_artifact: Optional[bool] = True*)

Uploads the model artifact to cloud storage.

**Parameters**

- **model\_artifact** (*Union[ModelArtifact, GenericModel]*) – The model artifacts or generic model instance.

- **provenance\_metadata** *((ModelProvenance, optional). Defaults to None.)* – Model provenance gives data scientists information about the origin of their model. This information allows data scientists to reproduce the development environment in which the model was trained.
- **project\_id** *((str, optional). Defaults to None.)* – The project\_id of model.
- **display\_name** *((str, optional). Defaults to None.)* – The name of model. If a display\_name is not provided, a randomly generated easy to remember name with times-tamp will be generated, like 'strange-spider-2022-08-17-23:55.02'.
- **description** *((str, optional). Defaults to None.)* – The description of model.
- **freeform\_tags** *((Dict[str, str], optional). Defaults to None.)* – Freeform tags for the model, by default None
- **defined\_tags** *((Dict[str, dict[str, object]], optional). Defaults to None.)* – Defined tags for the model, by default None.
- **bucket\_uri** *((str, optional). Defaults to None.)* – The OCI Object Storage URI where model artifacts will be copied to. The *bucket\_uri* is only necessary for uploading large artifacts which size greater than 2GB. Example: *oci://<bucket\_name>@<namespace>/prefix/*.
- **remove\_existing\_artifact** *((bool, optional). Defaults to True.)* – Whether artifacts uploaded to object storage bucket need to be removed or not.
- **overwrite\_existing\_artifact** *((bool, optional). Defaults to True.)* – Overwrite target bucket artifact if exists.

**Returns**

The ads.catalog.Model with the matching ID.

**Return type**

ads.catalog.Model

```
class ads.catalog.model.ModelSummaryList(model_catalog, model_list, response=None,
                                         datetime_format='%Y-%m-%d %H:%M:%S')
```

Bases: [SummaryList](#)

Model Summary List which represents a list of Model Object.

**sort\_by**(self, columns, reverse=False)

Performs a multi-key sort on a particular set of columns and returns the sorted ModelSummaryList. Results are listed in a descending order by default.

**filter**(self, selection, instance=None)

Filters the model list according to a lambda filter function, or list comprehension.

**filter**(selection, instance=None)

Filters the model list according to a lambda filter function, or list comprehension.

**Parameters**

- **selection** *(lambda function filtering model instances, or a list-comprehension)* – function of list filtering projects
- **instance** *(list, optional)* – list to filter, optional, defaults to self

**Returns**

ModelSummaryList



**Return type**

A filtered ModelSummaryList

**sort\_by**(*columns*, *reverse=False*)

Performs a multi-key sort on a particular set of columns and returns the sorted ModelSummaryList. Results are listed in a descending order by default.

**Parameters**

- **columns** (*List of string*) – A list of columns which are provided to sort on
- **reverse** (*Boolean (defaults to false)*) – If you'd like to reverse the results (for example, to get ascending instead of descending results)

**Returns**

ModelSummaryList

**Return type**

A sorted ModelSummaryList

**exception** `ads.catalog.model.ModelWithActiveDeploymentError`

Bases: Exception

**18.1.1.2.3 ads.catalog.notebook module**

**class** `ads.catalog.notebook.NotebookCatalog`(*compartment\_id=None*)

Bases: object

**create\_notebook\_session**(*display\_name=None*, *project\_id=None*, *shape=None*,  
*block\_storage\_size\_in\_gbs=None*, *subnet\_id=None*, *\*\*kwargs*)

Create a new notebook session with the supplied details.

**Parameters**

- **display\_name** (*str, required*) – The value to assign to the `display_name` property of this `CreateNotebookSessionDetails`.
- **project\_id** (*str, required*) – The value to assign to the `project_id` property of this `CreateNotebookSessionDetails`.
- **shape** (*str, required*) – The value to assign to the `shape` property of this `NotebookSessionConfigurationDetails`. Allowed values for this property are: “VM.Standard.E2.2”, “VM.Standard.E2.4”, “VM.Standard.E2.8”, “VM.Standard2.1”, “VM.Standard2.2”, “VM.Standard2.4”, “VM.Standard2.8”, “VM.Standard2.16”, “VM.Standard2.24”.
- **block\_storage\_size\_in\_gbs** (*int, required*) – Size of the block storage drive. Limited to values between 50 (GB) and 1024 (1024GB = 1TB)
- **subnet\_id** (*str, required*) – The OCID of the subnet resource where the notebook is to be created.
- **kwargs** (*dict, optional*) – Additional kwargs passed to `DataScienceClient.create_notebook_session()`

**Returns**

`oci.data_science.models.NotebookSession`

**Return type**

A new notebook record.

**Raises**

**KeyError** – If the resource was not found or do not have authorization to access that resource.:

**delete\_notebook\_session**(*notebook*, *\*\*kwargs*)

Deletes the notebook based on notebook\_id.

**Parameters**

**notebook** (*str ID or oci.data\_science.models.NotebookSession, required*) – The OCID of the notebook to delete as a string, or a Notebook Session instance

**Returns**

**Bool**

**Return type**

True if delete was successful, false otherwise

**get\_notebook\_session**(*notebook\_id*)

Get the notebook based on notebook\_id

**Parameters**

**notebook\_id** (*str, required*) – The OCID of the notebook to get.

**Returns**

**oci.data\_science.models.NotebookSession**

**Return type**

The oci.data\_science.models.NotebookSession with the matching ID.

**Raises**

**KeyError** – If the resource was not found or do not have authorization to access that resource.:

**list\_notebook\_session**(*include\_deleted=False, datetime\_format='%Y-%m-%d %H:%M:%S', \*\*kwargs*)

List all notebooks in a given compartment

**Parameters**

- **include\_deleted** (*bool, optional, default=False*) – Whether to include deleted notebooks in the returned list
- **datetime\_format** (*str, optional, default: '%Y-%m-%d %H:%M:%S'*) – Change format for date time fields

**Returns**

**NotebookSummaryList**

**Return type**

A List of notebooks.

**Raises**

**KeyError** – If the resource was not found or do not have authorization to access that resource.:

**update\_notebook\_session**(*notebook\_id, update\_notebook\_details=None, \*\*kwargs*)

Updates a notebook with given notebook\_id, using the provided update data

**Parameters**

- **notebook\_id** (*str*) – notebook\_id OCID to update
- **update\_notebook\_details** (*oci.data\_science.models.UpdateNotebookSessionDetails*) – contains the new notebook details data to apply
- **kwargs** (*dict, optional*) – Update notebook session details can be supplied instead as kwargs

**Returns****oci.data\_science.models.NotebookSession****Return type**

The updated Notebook record

**Raises****KeyError** – If the resource was not found or do not have authorization to access that resource.:

```
class ads.catalog.notebook.NotebookSummaryList(notebook_list, response=None,
                                                datetime_format='%Y-%m-%d %H:%M:%S')
```

Bases: *SummaryList***filter**(*selection*, *instance=None*)

Filter the notebook list according to a lambda filter function, or list comprehension.

**Parameters**

- **selection** (*lambda function filtering notebook instances, or a list-comprehension*) – function of list filtering notebooks
- **instance** (*list, optional*) – list to filter, optional, defaults to self

**Raises****ValueError** – If selection passed is not correct. For example: `selection=oci.data_science.models.NotebookSession.:`**sort\_by**(*columns*, *reverse=False*)

Performs a multi-key sort on a particular set of columns and returns the sorted NotebookSummaryList. Results are listed in a descending order by default.

**Parameters**

- **columns** (*List of string*) – A list of columns which are provided to sort on
- **reverse** (*Boolean (defaults to false)*) – If you'd like to reverse the results (for example, to get ascending instead of descending results)

**Returns****NotebookSummaryList****Return type**

A sorted NotebookSummaryList

**18.1.1.2.4 ads.catalog.project module**

```
class ads.catalog.project.ProjectCatalog(compartment_id=None, ds_client_auth=None,
                                         identity_client_auth=None)
```

Bases: Mapping

**create\_project**(*create\_project\_details=None*, *\*\*kwargs*)Create a new project with the supplied details. `create_project_details` contains parameters needed to create a new project, according to `oci.data_science.models.CreateProjectDetails`.**Parameters**

- **display\_name** (*str*) – The value to assign to the `display_name` property of this `CreateProjectDetails`.
- **description** (*str*) – The value to assign to the `description` property of this `CreateProjectDetails`.

- **compartment\_id** (*str*) – The value to assign to the compartment\_id property of this CreateProjectDetails.
- **freeform\_tags** (*dict(str, str)*) – The value to assign to the freeform\_tags property of this CreateProjectDetails.
- **defined\_tags** (*dict(str, dict(str, object))*) – The value to assign to the defined\_tags property of this CreateProjectDetails.
- **kwargs** – New project details can be supplied instead as kwargs

**Returns**

**oci.data\_science.models.Project**

**Return type**

A new Project record.

**delete\_project**(*project, \*\*kwargs*)

Deletes the project based on project\_id.

**Parameters**

**project** (*str ID or oci.data\_science.models.Project, required*) – The OCID of the project to delete as a string, or a Project instance

**Returns**

**Bool**

**Return type**

True if delete was succesful

**get\_project**(*project\_id*)

Get the Project based on project\_id

**Parameters**

**project\_id** (*str, required*) – The OCID of the project to get.

**Return type**

The oci.data\_science.models.Project with the matching ID.

**Raises**

**KeyError** – If the resource was not found or do not have authorization to access that resource.:

**list\_projects**(*include\_deleted=False, datetime\_format='%Y-%m-%d %H:%M:%S', \*\*kwargs*)

List all projects in a given compartment, or in the current notebook session's compartment

**Parameters**

- **include\_deleted** (*bool, optional, default=False*) – Whether to include deleted projects in the returned list
- **datetime\_format** (*str, optional, default: '%Y-%m-%d %H:%M:%S'*) – Change format for date time fields

**Returns**

**ProjectSummaryList**

**Return type**

List of Projects.

**Raises**

**KeyError** – If the resource was not found or do not have authorization to access that resource.:

**update\_project**(*project\_id*, *update\_project\_details=None*, *\*\*kwargs*)

Updates a project with given *project\_id*, using the provided update data *update\_project\_details* contains the update project details data to apply, according to `oci.data_science.models.UpdateProjectDetails`

#### Parameters

- **project\_id** (*str*) – project\_id OCID to update
- **display\_name** (*str*) – The value to assign to the `display_name` property of this `UpdateProjectDetails`.
- **description** (*str*) – The value to assign to the `description` property of this `UpdateProjectDetails`.
- **freeform\_tags** (*dict(str, str)*) – The value to assign to the `freeform_tags` property of this `UpdateProjectDetails`.
- **defined\_tags** (*dict(str, dict(str, object))*) – The value to assign to the `defined_tags` property of this `UpdateProjectDetails`.
- **kwargs** (*dict, optional*) – Update project details can be supplied instead as `kwargs`

#### Returns

`oci.data_science.models.Project`

#### Return type

The updated Project record

```
class ads.catalog.project.ProjectSummaryList(project_list, response=None,
                                             datetime_format='%Y-%m-%d %H:%M:%S')
```

Bases: `SummaryList`

A class used to represent Project Summary List.

...

**df**

Summary information for a project.

#### Type

data frame

**datetime\_format**

Format used to describe time.

#### Type

str

**response**

A response object with data of type list of `ProjectSummaryList`.

#### Type

`oci.response.Response`

**short\_id\_index**

Mapping of short id and its value.

#### Type

(dict of str: str)

**sort\_by(self, columns, reverse=False):**

Sort `ProjectSummaryList` by columns.

**filter(self, selection, instance=None):**

Filter the project list according to a lambda filter function, or list comprehension.

**filter(selection, instance=None)**

Filter the project list according to a lambda filter function, or list comprehension.

**Parameters**

- **selection** (*lambda function filtering Project instances, or a list-comprehension*) – function of list filtering projects
- **instance** (*list, optional*) – list to filter, optional, defaults to self

**Returns**

**ProjectSummaryList**

**Return type**

A filtered ProjectSummaryList

**Raises**

**ValueError** – If selection passed is not correct.:

**sort\_by(columns, reverse=False)**

Sort ProjectSummaryList by columns.

Performs a multi-key sort on a particular set of columns and returns the sorted ProjectSummaryList Results are listed in a descending order by default.

**Parameters**

- **columns** (*List of string*) – A list of columns which are provided to sort on
- **reverse** (*Boolean (defaults to false)*) – If you'd like to reverse the results (for example, to get ascending instead of descending results)

**Returns**

**ProjectSummaryList**

**Return type**

A sorted ProjectSummaryList

### 18.1.1.2.5 ads.catalog.summary module

**class** ads.catalog.summary.**SummaryList**(entity\_list, datetime\_format='%Y-%m-%d %H:%M:%S')

Bases: list

**abstract filter**(selection, instance=None)

Abstract method for filtering, implemented by the derived class

**show\_in\_notebook**(datetime\_format=None)

Displays the model catalog summary in a Jupyter Notebook cell

**Parameters**

**date\_format** (*like utils.date\_format. Defaults to none.*) –

**Return type**

None

**abstract sort\_by**(columns, reverse=False)

Abstract method for sorting, implemented by the derived class

**to\_dataframe**(*datetime\_format=None*)

Returns the model catalog summary as a pandas dataframe

**Parameters**

**datetime\_format** (*date\_format*) – A datetime format, like `utils.date_format`. Defaults to `none`.

**Returns**

**Dataframe**

**Return type**

The pandas DataFrame representation of the model catalog summary

### 18.1.1.2.6 Module contents

### 18.1.1.3 ads.common package

#### 18.1.1.3.1 Submodules

#### 18.1.1.3.2 ads.common.card\_identifier module

credit card patterns refer to [https://en.wikipedia.org/wiki/Payment\\_card\\_number#Issuer\\_identification\\_number\\_\(IIN\)](https://en.wikipedia.org/wiki/Payment_card_number#Issuer_identification_number_(IIN))  
Active and frequent card information American Express: 34, 37 Diners Club (US & Canada): 54,55 Discover Card: 6011, 622126 - 622925, 624000 - 626999, 628200 - 628899, 64, 65 Master Card: 2221-2720, 51–55 Visa: 4

**class** `ads.common.card_identifier.card_identify`

Bases: `object`

**identify\_issue\_network**(*card\_number*)

Returns the type of credit card based on its digits

**Parameters**

**card\_number** (*String*) –

**Returns**

**String**

**Return type**

A string corresponding to the kind of credit card.

#### 18.1.1.3.3 ads.common.auth module

**class** `ads.common.auth.APIKey`(*args: Optional[Dict] = None*)

Bases: `AuthSignerGenerator`

Creates api keys auth instance. This signer is intended to be used when signing requests for a given user - it requires that user's ID, their private key and certificate fingerprint. It prepares extra arguments necessary for creating clients for variety of OCI services.

Signer created based on args provided. If not provided current values of according arguments will be used from current global state from `AuthState` class.

**Parameters**

**args** (*dict*) – args that are required to create api key config and signer. Contains keys: `oci_config`, `oci_config_location`, `oci_key_profile`, `client_kwargs`.

- `oci_config` is a configuration dict that can be used to create clients
- `oci_config_location` - path to config file
- `oci_key_profile` - the profile to load from config file
- `client_kwargs` - optional parameters for OCI client creation in next steps

**create\_signer()** → Dict

Creates api keys configuration and signer with extra arguments necessary for creating clients. Signer constructed from the `oci_config` provided. If not `'oci_config'`, configuration will be constructed from `'oci_config_location'` and `'oci_key_profile'` in place.

## Resturns

### dict

Contains keys - `config`, `signer` and `client_kwargs`.

- `config` contains the configuration information
- `signer` contains the signer object created. It is instantiated from `signer_callable`, or

signer provided in args used, or instantiated in place - `client_kwargs` contains the *client\_kwargs* that was passed in as input parameter

## Examples

```
>>> signer_args = dict(  
>>>     client_kwargs=client_kwargs  
>>> )  
>>> signer_generator = AuthFactory().signerGenerator(AuthType.API_KEY)  
>>> signer_generator(signer_args).create_signer()
```

**class** `ads.common.auth.AuthFactory`

Bases: `object`

`AuthFactory` class which contains list of registered signers and alllows to register new signers. Check documentation for more signers: <https://docs.oracle.com/en-us/iaas/tools/python/latest/api/signing.html>.

**Current signers:**

- `APIKey`
- `ResourcePrincipal`
- `InstancePrincipal`

```
classes = {'api_key': <class 'ads.common.auth.APIKey'>, 'instance_principal': <class  
'ads.common.auth.InstancePrincipal'>, 'resource_principal': <class  
'ads.common.auth.ResourcePrincipal'>}
```

**classmethod** `register(signer_type: str, signer: Any) → None`

Registers a new signer.

### Parameters

- **signer\_type** (*str*) – Singer type to be registers
- **signer** (`RecordParser`) – A new Singer class to be registered.



**Returns**

Nothing.

**Return type**

None

**signerGenerator**(*iam\_type*: *Optional*[*str*] = 'api\_key')

Generates signer classes based of *iam\_type*, which specify one of auth methods: 'api\_key', 'resource\_principal' or 'instance\_principal'.

**Parameters**

**iam\_type** (*str*, *default* 'api\_key') – type of auth provided in IAM\_TYPE environment variable or set in parameters in `ads.set_auth()` method.

**Returns**

returns one of classes, which implements creation of signer of specified type

**Return type**

*APIKey* or *ResourcePrincipal* or *InstancePrincipal*

**Raises**

**ValueError** – If *iam\_type* is not supported.

**class** `ads.common.auth.AuthSignerGenerator`Bases: `object`

Abstract class for auth configuration and signer creation.

**create\_signer**()**class** `ads.common.auth.AuthState(*args, **kwargs)`Bases: `object`

Class stores state of variables specified for auth method, configuration, configuration file location, profile name, signer or signer\_callable, which set by use at any given time and can be provided by this class in any ADS module.

**oci\_cli\_auth**: `str` = `None`**oci\_config**: `str` = `None`**oci\_config\_path**: `str` = `None`**oci\_iam\_type**: `str` = `None`**oci\_key\_profile**: `str` = `None`**oci\_signer**: `str` = `None`**oci\_signer\_callable**: `str` = `None`**oci\_signer\_kwargs**: `str` = `None`**class** `ads.common.auth.AuthType`Bases: `str`**API\_KEY** = 'api\_key'**INSTANCE\_PRINCIPAL** = 'instance\_principal'**RESOURCE\_PRINCIPAL** = 'resource\_principal'

**class** `ads.common.auth.InstancePrincipal`(*args: Optional[Dict] = None*)

Bases: [`AuthSignerGenerator`](#)

Creates Instance Principal signer - a `SecurityTokenSigner` which uses a security token for an instance principal. It prepares extra arguments necessary for creating clients for variety of OCI services.

Signer created based on args provided. If not provided current values of according arguments will be used from current global state from `AuthState` class.

#### Parameters

**args** (*dict*) – args that are required to create Instance Principal signer. Contains keys: `signer_kwargs`, `client_kwargs`.

- `signer_kwargs` - optional parameters required to instantiate instance principal signer
- `client_kwargs` - optional parameters for OCI client creation in next steps

**create\_signer()** → `Dict`

Creates Instance Principal signer with extra arguments necessary for creating clients. Signer instantiated from the *signer\_callable* or if the *signer* provided is will be return by this method. If *signer\_callable* or *signer* not provided new signer will be created in place.

#### Returns

**dict**

Contains keys - `config`, `signer` and `client_kwargs`.

- `config` contains the configuration information
- `signer` contains the signer object created. It is instantiated from *signer\_callable*, or

*signer* provided in args used, or instantiated in place - `client_kwargs` contains the *client\_kwargs* that was passed in as input parameter

#### Examples

```
>>> signer_args = dict(signer_kwargs={"log_requests": True})
>>> signer_generator = AuthFactory().signerGenerator(AuthType.INSTANCE_
↳ PRINCIPAL)
>>> signer_generator(signer_args).create_signer()
```

**class** `ads.common.auth.OCIAuthContext`(*profile: Optional[str] = None*)

Bases: `object`

`OCIAuthContext` used in 'with' statement for properly managing global authentication type and global configuration profile parameters.

## Examples

```
>>> from ads.jobs import DataFlowRun
>>> with OCIAuthContext(profile='TEST'):
>>>     df_run = DataFlowRun.from_ocid(run_id)
```

Initialize class OCIAuthContext and saves global state of authentication type and configuration profile.

### Parameters

**profile** (*str*, default is *None*) – profile name for api keys config file

**class** `ads.common.auth.ResourcePrincipal`(*args: Optional[Dict] = None*)

Bases: *AuthSignerGenerator*

Creates Resource Principal signer - a security token for a resource principal. It prepares extra arguments necessary for creating clients for variety of OCI services.

Signer created based on args provided. If not provided current values of according arguments will be used from current global state from AuthState class.

### Parameters

**args** (*dict*) – args that are required to create Resource Principal signer. Contains keys: *client\_kwargs*.

- *client\_kwargs* - optional parameters for OCI client creation in next steps

**create\_signer()** → *Dict*

Creates Resource Principal signer with extra arguments necessary for creating clients.

## Returns

### dict

Contains keys - *config*, *signer* and *client\_kwargs*.

- *config* contains the configuration information
- *signer* contains the signer object created. It is instantiated from *signer\_callable*, or

*signer* provided in *args* used, or instantiated in place - *client\_kwargs* contains the *client\_kwargs* that was passed in as input parameter

## Examples

```
>>> signer_args = dict(
>>>     signer=oci.auth.signers.get_resource_principals_signer()
>>> )
>>> signer_generator = AuthFactory().signerGenerator(AuthType.RESOURCE_
↳ PRINCIPAL)
>>> signer_generator(signer_args).create_signer()
```

**class** `ads.common.auth.SingletonMeta`

Bases: *type*

`ads.common.auth.api_keys`(*oci\_config: str = '/home/docs/.oci/config'*, *profile: str = 'DEFAULT'*, *client\_kwargs: Optional[Dict] = None*) → *Dict*

Prepares authentication and extra arguments necessary for creating clients for different OCI services using API Keys.

#### Parameters

- **oci\_config** (*Optional[str]*, default is `$HOME/.oci/config`) – OCI authentication config file location.
- **profile** (*Optional[str]*, is `DEFAULT_PROFILE`, which is `'DEFAULT'`) – Profile name to select from the config file.
- **client\_kwargs** (*Optional[Dict]*, default `None`) – kwargs that are required to instantiate the Client if we need to override the defaults.

#### Returns

Contains keys - config, signer and client\_kwargs.

- The config contains the config loaded from the configuration loaded from `oci_config`.
- The signer contains the signer object created from the api keys.
- client\_kwargs contains the `client_kwargs` that was passed in as input parameter.

#### Return type

dict

#### Examples

```
>>> from ads.common import oci_client as oc
>>> auth = ads.auth.api_keys(oci_config="/home/datascience/.oci/config", profile=
↳ "TEST", client_kwargs={"timeout": 6000})
>>> oc.OCIClientFactory(**auth).object_storage # Creates Object storage client with
↳ timeout set to 6000 using API Key authentication
```

```
ads.common.auth.create_signer(auth_type: Optional[str] = 'api_key', oci_config_location: Optional[str] =
'~/.oci/config', profile: Optional[str] = 'DEFAULT', config: Optional[Dict] =
{}, signer: Optional[Any] = None, signer_callable: Optional[Callable] =
None, signer_kwargs: Optional[Dict] = {}, client_kwargs: Optional[Dict] =
None) → Dict
```

Prepares authentication and extra arguments necessary for creating clients for different OCI services based on provided parameters. If `signer` or `signer_callable` provided, authentication with that signer will be created. If `config` provided, `api_key` type of authentication will be created. Accepted values for `auth_type`: `api_key` (default), `'instance_principal'`, `'resource_principal'`.

#### Parameters

- **auth** (*Optional[str]*, default `'api_key'`) – `'api_key'`, `'resource_principal'` or `'instance_principal'`. Enable/disable resource principal identity, instance principal or keypair identity in a notebook session
- **oci\_config\_location** (*Optional[str]*, default `oci.config.DEFAULT_LOCATION`, which is `'~/.oci/config'`) – config file location
- **profile** (*Optional[str]*, default is `DEFAULT_PROFILE`, which is `'DEFAULT'`) – profile name for api keys config file
- **config** (*Optional[Dict]*, default `{}`) – created config dictionary

- **signer** (*Optional[Any], default None*) – created signer, can be resource principals signer, instance principal signer or other. Check documentation for more signers: <https://docs.oracle.com/en-us/iaas/tools/python/latest/api/signing.html>
- **signer\_callable** (*Optional[Callable], default None*) – a callable object that returns signer
- **signer\_kwargs** (*Optional[Dict], default None*) – parameters accepted by the signer. Check documentation: <https://docs.oracle.com/en-us/iaas/tools/python/latest/api/signing.html>
- **client\_kwargs** (*dict*) – kwargs that are required to instantiate the Client if we need to override the defaults

## Examples

```
>>> import ads
>>> auth = ads.auth.create_signer() # api_key type of authentication dictionary
↳ created with default config location and default profile
```

```
>>> config = oci.config.from_file("other_config_location", "OTHER_PROFILE")
>>> auth = ads.auth.create_signer(config=config) # api_key type of authentication
↳ dictionary created based on provided config
```

```
>>> singer = oci.auth.signers.get_resource_principals_signer()
>>> auth = ads.auth.create_signer(config={}, singer=singer) # resource principals
↳ authentication dictionary created
```

```
>>> auth = ads.auth.create_signer(auth_type='instance_principal') # instance
↳ principals authentication dictionary created
```

```
>>> signer_callable = oci.auth.signers.InstancePrincipalsSecurityTokenSigner
>>> signer_kwargs = dict(log_requests=True) # will log the request url and response
↳ data when retrieving
>>> auth = ads.auth.create_signer(signer_callable=signer_callable, signer_
↳ kwargs=signer_kwargs) # instance principals authentication dictionary created
↳ based on callable with kwargs parameters
```

`ads.common.auth.default_signer(client_kwargs: Optional[Dict] = None) → Dict`

Prepares authentication and extra arguments necessary for creating clients for different OCI services based on the default authentication setting for the session. Refer `ads.set_auth` API for further reference.

### Parameters

**client\_kwargs** (*dict*) – kwargs that are required to instantiate the Client if we need to override the defaults.

### Returns

Contains keys - config, signer and client\_kwargs.

- The config contains the config loaded from the configuration loaded from the default location if the default auth mode is API keys, otherwise it is empty dictionary.
- The signer contains the signer object created from default auth mode.
- client\_kwargs contains the *client\_kwargs* that was passed in as input parameter.

**Return type**  
dict

### Examples

```
>>> import ads
>>> from ads.common import oci_client as oc
```

```
>>> auth = ads.auth.default_signer()
>>> oc.OCIClientFactory(**auth).object_storage # Creates Object storage client
```

```
>>> ads.set_auth("resource_principal")
>>> auth = ads.auth.default_signer()
>>> oc.OCIClientFactory(**auth).object_storage # Creates Object storage client,
↪ using resource principal authentication
```

```
>>> signer_callable = oci.auth.signers.InstancePrincipalsSecurityTokenSigner
>>> ads.set_auth(signer_callable=signer_callable) # Set instance principal callable
>>> auth = ads.auth.default_signer() # signer_callable instantiated
>>> oc.OCIClientFactory(**auth).object_storage # Creates Object storage client,
↪ using instance principal authentication
```

`ads.common.auth.get_signer(oci_config: Optional[str] = None, oci_profile: Optional[str] = None, **client_kwargs) → Dict`

Provides config and signer based given parameters. If `oci_config` (api key config file location) and `oci_profile` specified new signer will be generated. Else signer of a type specified in `OCI_CLI_AUTH` environment variable will be used to generate signer and return. If `OCI_CLI_AUTH` not set, resource principal signer will be provided. Accepted values for `OCI_CLI_AUTH`: 'api\_key', 'instance\_principal', 'resource\_principal'.

#### Parameters

- **oci\_config** (*Optional[str]*, *default None*) – Path to the config file
- **oci\_profile** (*Optional[str]*, *default None*) – the profile to load from the config file
- **client\_kwargs** – kwargs that are required to instantiate the Client if we need to override the defaults

`ads.common.auth.resource_principal(client_kwargs: Optional[Dict] = None) → Dict`

Prepares authentication and extra arguments necessary for creating clients for different OCI services using Resource Principals.

#### Parameters

**client\_kwargs** (*Dict*, *default None*) – kwargs that are required to instantiate the Client if we need to override the defaults.

#### Returns

Contains keys - config, signer and client\_kwargs.

- The config contains an empty dictionary.
- The signer contains the signer object created from the resource principal.
- client\_kwargs contains the *client\_kwargs* that was passed in as input parameter.

**Return type**  
dict

## Examples

```
>>> from ads.common import oci_client as oc
>>> auth = ads.auth.resource_principal({"timeout": 6000})
>>> oc.OCIClientFactory(**auth).object_storage # Creates Object Storage client with
↳ timeout set to 6000 seconds using resource principal authentication
```

```
ads.common.auth.set_auth(auth: Optional[str] = 'api_key', oci_config_location: Optional[str] = '~/.oci/config',
                          profile: Optional[str] = 'DEFAULT', config: Optional[Dict] = {}, signer:
                          Optional[Any] = None, signer_callable: Optional[Callable] = None,
                          signer_kwargs: Optional[Dict] = {}) → None
```

Save type of authentication, profile, config location, config (keypair identity) or signer, which will be used when actual creation of config or signer happens.

### Parameters

- **auth** (*Optional[str]*, default 'api\_key') –  
 'api\_key', 'resource\_principal' or 'instance\_principal'. Enable/disable resource principal identity,  
 instance principal or keypair identity in a notebook session
- **oci\_config\_location** (*Optional[str]*, default *oci.config.DEFAULT\_LOCATION*, which is '~/.oci/config') – config file location
- **profile** (*Optional[str]*, default is *DEFAULT\_PROFILE*, which is 'DEFAULT') – profile name for api keys config file
- **config** (*Optional[Dict]*, default {}) – created config dictionary
- **signer** (*Optional[Any]*, default *None*) – created signer, can be resource principals signer, instance principal signer or other. Check documentation for more signers: <https://docs.oracle.com/en-us/iaas/tools/python/latest/api/signing.html>
- **signer\_callable** (*Optional[Callable]*, default *None*) – a callable object that returns signer
- **signer\_kwargs** (*Optional[Dict]*, default *None*) – parameters accepted by the signer. Check documentation: <https://docs.oracle.com/en-us/iaas/tools/python/latest/api/signing.html>

## Examples

```
>>> ads.set_auth("api_key") # default signer is set to api keys
```

```
>>> ads.set_auth("api_key", profile = "TEST") # default signer is set to api keys
↳ and to use TEST profile
```

```
>>> ads.set_auth("api_key", oci_config_location = "other_config_location") # use
↳ non-default oci_config_location
```

```
>>> other_config = oci.config.from_file("other_config_location", "OTHER_PROFILE") #
↳ Create non-default config
>>> ads.set_auth(config=other_config) # Set api keys type of authentication based
↳ on provided config
```

```
>>> ads.set_auth("resource_principal") # Set resource principal authentication
```

```
>>> ads.set_auth("instance_principal") # Set instance principal authentication
```

```
>>> singer = oci.auth.signers.get_resource_principals_signer()
>>> ads.auth.create_signer(config={}, singer=singer) # resource principals_
↪ authentication dictionary created
```

```
>>> singer_callable = oci.auth.signers.ResourcePrincipalsFederationSigner
>>> ads.set_auth(singer_callable=singer_callable) # Set resource principal_
↪ federation singer callable
```

```
>>> singer_callable = oci.auth.signers.InstancePrincipalsSecurityTokenSigner
>>> singer_kwargs = dict(log_requests=True) # will log the request url and response_
↪ data when retrieving
>>> # instance principals authentication dictionary created based on callable with_
↪ kwargs parameters:
>>> ads.set_auth(singer_callable=singer_callable, singer_kwargs=singer_kwargs)
```

#### 18.1.1.3.4 ads.common.data module

**class** ads.common.data.ADSData(X=None, y=None, name="", dataset\_type=None)

Bases: object

This class wraps the input dataframe to various models, evaluation, and explanation frameworks. It's primary purpose is to hold any metadata relevant to these tasks. This can include it's:

- X - the independent variables as some dataframe-like structure,
- y - the dependent variable or target column as some array-like structure,
- name - a string to name the data for user convenience,
- dataset\_type - the type of the X value.

As part of this initiative, ADSData knows how to turn itself into an onnxruntime compatible data structure with the method `.to_onnxrt()`, which takes an onnx session as input.

##### Parameters

- **X** (Union[pandas.DataFrame, dask.DataFrame, numpy.ndarray, scipy.sparse.csr.csr\_matrix]) – If str, URI for the dataset. The dataset could be read from local or network file system, hdfs, s3 and gcs. Should be none if X\_train, y\_train, X\_test, Y\_test are provided
- **y** (Union[str, pandas.DataFrame, dask.DataFrame, pandas.Series, dask.Series, numpy.ndarray]) – If str, name of the target in X, otherwise series of labels corresponding to X
- **name** (str, optional) – Name to identify this data
- **dataset\_type** (ADSDataset optional) – When this value is available, would be used to evaluate the ads task type
- **kwargs** – Additional keyword arguments that would be passed to the underlying Pandas read API.



**static build**(*X=None, y=None, name="", dataset\_type=None, \*\*kwargs*)

Returns an ADSDataset object built from the (source, target) or (X,y)

#### Parameters

- **X** (*Union[pandas.DataFrame, dask.DataFrame, numpy.ndarray, scipy.sparse.csr.csr\_matrix]*) – If str, URI for the dataset. The dataset could be read from local or network file system, hdfs, s3 and gcs. Should be none if X\_train, y\_train, X\_test, Y\_test are provided
- **y** (*Union[str, pandas.DataFrame, dask.DataFrame, pandas.Series, dask.Series, numpy.ndarray]*) – If str, name of the target in X, otherwise series of labels corresponding to X
- **name** (*str, optional*) – Name to identify this data
- **dataset\_type** (*ADSDataset, optional*) – When this value is available, would be used to evaluate the ads task type
- **kwargs** – Additional keyword arguments that would be passed to the underlying Pandas read API.

#### Returns

**ads\_data** – A built ADSDataset object

#### Return type

*ads.common.data.ADSData*

### Examples

```
>>> data = open_csv("my.csv")
```

```
>>> data_ads = ADSDataset(data, 'target').build(data, 'target')
```

**to\_onnxrt**(*sess, idx\_range=None, model=None, impute\_values={}, \*\*kwargs*)

Returns itself formatted as an input for the onnxruntime session inputs passed in.

#### Parameters

- **sess** (*Session*) – The session object
- **idx\_range** (*Range*) – The range of inputs to convert to onnx
- **model** (*SupportedModel*) – A model that supports being serialized for the onnx runtime.
- **kwargs** (*additional keyword arguments*) –
  - **sess\_inputs** – Pass in the output from onnxruntime.InferenceSession(“model.onnx”).get\_inputs()
  - **input\_dtypes** (list) - If sess\_inputs cannot be passed in, pass in the numpy dtypes of each input
  - **input\_shapes** (list) - If sess\_inputs cannot be passed in, pass in the shape of each input
  - **input\_names** (list) - If sess\_inputs cannot be passed in, pass in the name of each input

#### Returns

**ort** – array of inputs formatted for the given session.

#### Return type

Array

### 18.1.1.3.5 ads.common.model module

**class** ads.common.model.ADSModel(*est*, *target=None*, *transformer\_pipeline=None*, *client=None*,  
*booster=None*, *classes=None*, *name=None*)

Bases: object

Construct an ADSModel

#### Parameters

- **est** (*fitted estimator object*) – The estimator can be a standard sklearn estimator, a keras, lightgbm, or xgboost estimator, or any other object that implement methods from (BaseEstimator, RegressorMixin) for regression or (BaseEstimator, ClassifierMixin) for classification.
- **target** (*PandasSeries*) – The target column you are using in your dataset, this is assigned as the “y” attribute.
- **transformer\_pipeline** (*TransformerPipeline*) – A custom transformer pipeline object.
- **client** (*Str*) – Currently unused.
- **booster** (*Str*) – Currently unused.
- **classes** (*list, optional*) – List of target classes. Required for classification problem if the est does not contain *classes* attribute.
- **name** (*str, optional*) – Name of the model.

**static** convert\_dataframe\_schema(*df*, *drop=None*)

**feature\_names**(*X=None*)

**static** from\_estimator(*est*, *transformers=None*, *classes=None*, *name=None*)

Build ADSModel from a fitted estimator

#### Parameters

- **est** (*fitted estimator object*) – The estimator can be a standard sklearn estimator or any object that implement methods from (BaseEstimator, RegressorMixin) for regression or (BaseEstimator, ClassifierMixin) for classification.
- **transformers** (*a scalar or an iterable of objects implementing transform function, optional*) – The transform function would be applied on data before calling predict and predict\_proba on estimator.
- **classes** (*list, optional*) – List of target classes. Required for classification problem if the est does not contain *classes* attribute.
- **name** (*str, optional*) – Name of the model.

**Returns**  
**model**

**Return type**

*ads.common.model.ADSModel*

## Examples

```
>>> model = MyModelClass.train()
>>> model_ads = from_estimator(model)
```

**static** `get_init_types(df, underlying_model=None)`

**is\_classifier()**

Returns True if ADS believes that the model is a classifier

**Returns**

**Boolean**

**Return type**

True if the model is a classifier, False otherwise.

**predict(X)**

Runs the models predict function on some data

**Parameters**

**X** ([ADSDData](#)) – A ADSData object which holds the examples to be predicted on.

**Returns**

Usually a list or PandasSeries of predictions

**Return type**

Union[List, pandas.Series], depending on the estimator

**predict\_proba(X)**

Runs the models predict probabilities function on some data

**Parameters**

**X** ([ADSDData](#)) – A ADSData object which holds the examples to be predicted on.

**Returns**

Usually a list or PandasSeries of predictions

**Return type**

Union[List, pandas.Series], depending on the estimator

**prepare**(*target\_dir=None, data\_sample=None, X\_sample=None, y\_sample=None, include\_data\_sample=False, force\_overwrite=False, fn\_artifact\_files\_included=False, fn\_name='model\_api', inference\_conda\_env=None, data\_science\_env=False, ignore\_deployment\_error=False, use\_case\_type=None, inference\_python\_version=None, imputed\_values={}, \*\*kwargs*)

Prepare model artifact directory to be published to model catalog

**Parameters**

- **target\_dir** (*str, default: model.name[:12]*) – Target directory under which the model artifact files need to be added
- **data\_sample** ([ADSDData](#)) – Note: This format is preferable to X\_sample and y\_sample. A sample of the test data that will be provided to predict() API of scoring script Used to generate schema\_input.json and schema\_output.json which defines the input and output formats
- **X\_sample** (*pandas.DataFrame*) – A sample of input data that will be provided to predict() API of scoring script Used to generate schema.json which defines the input formats

- **y\_sample** (*pandas.Series*) – A sample of output data that is expected to be returned by predict() API of scoring script, corresponding to X\_sample Used to generate schema\_output.json which defines the output formats
- **force\_overwrite** (*bool*, *default: False*) – If True, overwrites the target directory if exists already
- **fn\_artifact\_files\_included** (*bool*, *default: True*) – If True, generates artifacts to export a model as a function without ads dependency
- **fn\_name** (*str*, *default: 'model\_api'*) – Required parameter if fn\_artifact\_files\_included parameter is setup.
- **inference\_conda\_env** (*str*, *default: None*) – Conda environment to use within the model deployment service for inferencing
- **data\_science\_env** (*bool*, *default: False*) – If set to True, datascience environment represented by the slug in the training conda environment will be used.
- **ignore\_deployment\_error** (*bool*, *default: False*) – If set to True, the prepare will ignore all the errors that may impact model deployment
- **use\_case\_type** (*str*) – The use case type of the model. Use it through UseCaseType class or string provided in UseCaseType. For example, use\_case\_type=UseCaseType.BINARY\_CLASSIFICATION or use\_case\_type="binary\_classification". Check with UseCaseType class to see all supported types.
- **inference\_python\_version** (*str*, *default: None*.) – If provided will be added to the generated runtime yaml
- **\*\*kwargs** –
- -----
- **max\_col\_num** (*((int, optional). Defaults to utils.DATA\_SCHEMA\_MAX\_COL\_NUM.*) – The maximum column size of the data that allows to auto generate schema.

#### Returns

**model\_artifact**

#### Return type

an instance of *ModelArtifact* that can be used to test the generated scoring script

**rename**(*name*)

Changes the name of a model

#### Parameters

**name** (*str*) – A string which is supplied for naming a model.

**score**(*X*, *y\_true*, *score\_fn=None*)

Scores a model according to a custom score function

#### Parameters

- **X** (*ADSDData*) – A ADSData object which holds the examples to be predicted on.
- **y\_true** (*ADSDData*) – A ADSData object which holds ground truth labels for the examples which are being predicted on.
- **score\_fn** (*Scorer (callable)*) – A callable object that returns a score, usually created with sklearn.metrics.make\_scorer().

**Returns**

Almost always a scalar score (usually a float).

**Return type**

float, depending on the estimator

**show\_in\_notebook()**

Describe the model by showing it's properties

**summary()**

A summary of the ADSModel

**transform(X)**

Process some ADSData through the selected ADSModel transformers

**Parameters**

**X** (*ADSData*) – A ADSData object which holds the examples to be transformed.

**visualize\_transforms()**

A graph of the ADSModel transformer pipeline. It is only supported in JupyterLabs Notebooks.

**18.1.1.3.6 ads.common.model\_metadata module**

The module created for the back compatability. The original *model\_metadata* was moved to the *ads.model* package.

**18.1.1.3.7 ads.common.decorator.runtime\_dependency module**

The module that provides the decorator helping to add runtime dependencies in functions.

**Examples**

```
>>> @runtime_dependency(module="pandas", short_name="pd")
... def test_function()
...     print(pd)
```

```
>>> @runtime_dependency(module="pandas", object="DataFrame", short_name="df")
... def test_function()
...     print(df)
```

```
>>> @runtime_dependency(module="pandas", short_name="pd")
... @runtime_dependency(module="pandas", object="DataFrame", short_name="df")
... def test_function()
...     print(df)
...     print(pd)
```

```
>>> @runtime_dependency(module="pandas", object="DataFrame", short_name="df", install_
↳ from="ads[optional]")
... def test_function()
...     pass
```

```
>>> @runtime_dependency(module="pandas", object="DataFrame", short_name="df", err_msg=
↳ "Custom error message.")
... def test_function()
...     pass
```

```
class ads.common.decorator.runtime_dependency.OptionalDependency
```

```
    Bases: object
```

```
    BDS = 'oracle-ads[bds]'
```

```
    BOOSTED = 'oracle-ads[boosted]'
```

```
    DATA = 'oracle-ads[data]'
```

```
    GEO = 'oracle-ads[geo]'
```

```
    LABS = 'oracle-ads[labs]'
```

```
    MYSQL = 'oracle-ads[mysql]'
```

```
    NOTEBOOK = 'oracle-ads[notebook]'
```

```
    ONNX = 'oracle-ads[onnx]'
```

```
    OPCTL = 'oracle-ads[opctl]'
```

```
    OPTUNA = 'oracle-ads[optuna]'
```

```
    PYTORCH = 'oracle-ads[torch]'
```

```
    SPARK = 'oracle-ads[spark]'
```

```
    TENSORFLOW = 'oracle-ads[tensorflow]'
```

```
    TEXT = 'oracle-ads[text]'
```

```
    VIZ = 'oracle-ads[viz]'
```

```
ads.common.decorator.runtime_dependency.runtime_dependency(module: str, short_name: str = "",
                                                           object: Optional[str] = None,
                                                           install_from: Optional[str] = None,
                                                           err_msg: str = "",
                                                           is_for_notebook_only=False)
```

The decorator which is helping to add runtime dependencies to functions.

#### Parameters

- **module** (*str*) – The module name to be imported.
- **short\_name** (*((str, optional). Defaults to empty string.)*) – The short name for the imported module.
- **object** (*((str, optional). Defaults to None.)*) – The name of the object to be imported. Can be a function or a class, or any variable provided by module.
- **install\_from** (*((str, optional). Defaults to None.)*) – The parameter helping to answer from where the required dependency can be installed.
- **err\_msg** (*((str, optional). Defaults to empty string.)*) – The custom error message.

- **is\_for\_notebook\_only** ((*bool*, *optional*). Defaults to *False*.) – If the value of this flag is set to *True*, the dependency will be added only in case when the current environment is a jupyter notebook.

#### Raises

- **ModuleNotFoundError** – In case if requested module not found.
- **ImportError** – In case if object cannot be imported from the module.

#### Examples

```
>>> @runtime_dependency(module="pandas", short_name="pd")
... def test_function()
...     print(pd)
```

```
>>> @runtime_dependency(module="pandas", object="DataFrame", short_name="df")
... def test_function()
...     print(df)
```

```
>>> @runtime_dependency(module="pandas", short_name="pd")
... @runtime_dependency(module="pandas", object="DataFrame", short_name="df")
... def test_function()
...     print(df)
...     print(pd)
```

```
>>> @runtime_dependency(module="pandas", object="DataFrame", short_name="df",
↳ install_from="ads[optional]")
... def test_function()
...     pass
```

```
>>> @runtime_dependency(module="pandas", object="DataFrame", short_name="df", err_
↳ msg="Custom error message.")
... def test_function()
...     pass
```

#### 18.1.1.3.8 ads.common.decorator.deprecate module

**class** ads.common.decorator.deprecate.TARGET\_TYPE(*value*)

Bases: Enum

An enumeration.

**ATTRIBUTE** = 'Attribute'

**CLASS** = 'Class'

**METHOD** = 'Method'

ads.common.decorator.deprecate.**deprecated**(*deprecated\_in*: str, *removed\_in*: Optional[str] = None, *details*: Optional[str] = None, *target\_type*: Optional[str] = None)

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

#### Parameters

- **deprecated\_in** (*str*) – Version of ADS where this function deprecated.
- **removed\_in** (*str*) – Future version where this function will be removed.
- **details** (*str*) – More information to be shown.

### 18.1.1.3.9 ads.common.model\_introspect module

The module that helps to minimize the number of errors of the model post-deployment process. The model provides a simple testing harness to ensure that model artifacts are thoroughly tested before being saved to the model catalog.

#### Classes

##### ModelIntrospect

Class to introspect model artifacts.

#### Examples

```
>>> model_introspect = ModelIntrospect(artifact=model_artifact)
>>> model_introspect()
... Test key      Test name      Result      Message
... -----
... test_key_1    test_name_1    Passed      test passed
... test_key_2    test_name_2    Not passed  some error occurred
>>> model_introspect.status
... Passed
```

```
class ads.common.model_introspect.Introspectable
```

Bases: ABC

Base class that represents an introspectable object.

```
exception ads.common.model_introspect.IntrospectionNotPassed
```

Bases: ValueError

```
class ads.common.model_introspect.ModelIntrospect(artifact: Introspectable)
```

Bases: object

Class to introspect model artifacts.

#### Parameters

- **status** (*str*) – Returns the current status of model introspection. The possible variants: *Passed, Not passed, Not tested*.
- **failures** (*int*) – Returns the number of failures of introspection result.

**run**(*self*) → None

Invokes model artifacts introspection.



**to\_dataframe(self)** → pd.DataFrame

Serializes model introspection result into a DataFrame.

## Examples

```
>>> model_introspect = ModelIntrospect(artifact=model_artifact)
>>> result = model_introspect()
... Test key          Test name          Result          Message
... -----
... test_key_1        test_name_1        Passed           test passed
... test_key_2        test_name_2        Not passed       some error occurred
```

Initializes the Model Introspect.

### Parameters

**artifact** ([Introspectable](#)) – The instance of ModelArtifact object.

### Raises

- **ValueError** – If model artifact object not provided.:
- **TypeError** – If provided input paramater not a ModelArtifact instance.:

**property failures: int**

Calculates the number of failures.

### Returns

The number of failures.

### Return type

int

**run()** → DataFrame

Invokes introspection.

### Returns

The introspection result in a DataFrame format.

### Return type

pd.DataFrame

**property status: str**

Gets the current status of model introspection.

**to\_dataframe()** → DataFrame

Serializes model introspection result into a DataFrame.

### Returns

The model introspection result in a DataFrame representation.

### Return type

*pandas.DataFrame*

**class ads.common.model\_introspect.PrintItem**(key: str = "", case: str = "", result: str = "", message: str = "")

Bases: object

Class represents the model introspection print item.

**case: str = ''**

```
key: str = ''
```

```
message: str = ''
```

```
result: str = ''
```

```
to_list() → List[str]
```

Converts instance to a list representation.

#### Returns

The instance in a list representation.

#### Return type

List[str]

```
class ads.common.model_introspect.TEST_STATUS
```

```
    Bases: str
```

```
    NOT_PASSED = 'Failed'
```

```
    NOT_TESTED = 'Skipped'
```

```
    PASSED = 'Passed'
```

### 18.1.1.3.10 ads.common.model\_export\_util module

```
ads.common.model_export_util.prepare_generic_model(model_path: str, fn_artifact_files_included: bool = False, fn_name: str = 'model_api', force_overwrite: bool = False, model: Any = None, data_sample: ADSData = None, use_case_type=None, X_sample: Union[list, tuple, Series, ndarray, DataFrame] = None, y_sample: Union[list, tuple, Series, ndarray, DataFrame] = None, **kwargs) → ModelArtifact
```

Generates template files to aid model deployment. The model could be accompanied by other artifacts all of which can be dumped at *model\_path*. Following files are generated: \* func.yaml \* func.py \* requirements.txt \* score.py

#### Parameters

- **model\_path** (*str*) – Path where the artifacts must be saved. The serialized model object and any other associated files/objects must be saved in the *model\_path* directory
- **fn\_artifact\_files\_included** (*bool*) – Default is False, if turned off, function artifacts are not generated.
- **fn\_name** (*str*) – Optional parameter to specify the function name
- **force\_overwrite** (*bool*) – Optional parameter to specify if the model\_artifact should overwrite the existing model\_path (if it exists)
- **model** (*(Any, optional). Defaults to None.*) – This is an optional model object which is only used to extract taxonomy metadata. Supported models: automl, keras, lightgbm, pytorch, sklearn, tensorflow, and xgboost. If the model is not under supported frameworks, then extracting taxonomy metadata will be skipped. The alternative way is using *artifact.populate\_metadata(model=model, usecase\_type=UseCaseType.REGRESSION)*.
- **data\_sample** (*ADSData*) – A sample of the test data that will be provided to predict() API of scoring script Used to generate schema\_input and schema\_output

- **use\_case\_type** (*str*) – The use case type of the model
- **X\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame, dask.dataframe.core.Series, dask.dataframe.core.DataFrame]*) – A sample of input data that will be provided to predict() API of scoring script Used to generate input schema.
- **y\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame, dask.dataframe.core.Series, dask.dataframe.core.DataFrame]*) – A sample of output data that is expected to be returned by predict() API of scoring script, corresponding to X\_sample Used to generate output schema.
- **\*\*kwargs** –
- **\_\_\_\_\_** –
- **data\_science\_env** (*bool, default: False*) – If set to True, the datascience environment represented by the slug in the training conda environment will be used.
- **inference\_conda\_env** (*str, default: None*) – Conda environment to use within the model deployment service for inferencing. For example, `oci://bucketname@namespace/path/to/conda/env`
- **ignore\_deployment\_error** (*bool, default: False*) – If set to True, the prepare method will ignore all the errors that may impact model deployment.
- **underlying\_model** (*str, default: 'UNKNOWN'*) – Underlying Model Type, could be “automl”, “sklearn”, “h2o”, “lightgbm”, “xgboost”, “torch”, “mxnet”, “tensorflow”, “keras”, “pyod” and etc.
- **model\_libs** (*dict, default: {}*) – Model required libraries where the key is the library names and the value is the library versions. For example, {numpy: 1.21.1}.
- **progress** (*int, default: None*) – max number of progress.
- **inference\_python\_version** (*str, default: None.*) – If provided will be added to the generated runtime yaml
- **max\_col\_num** (*((int, optional). Defaults to utils.DATA\_SCHEMA\_MAX\_COL\_NUM.)*) – The maximum column size of the data that allows to auto generate schema.

## Examples

```
>>> import cloudpickle
>>> import os
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import make_classification
>>> import ads
>>> from ads.common.model_export_util import prepare_generic_model
>>> import yaml
>>> import oci
>>>
>>> ads.set_auth('api_key', oci_config_location=oci.config.DEFAULT_LOCATION,
↳profile='DEFAULT')
>>> model_artifact_location = os.path.expanduser('~/.myusecase/model/')
>>> inference_conda_env="oci://my-bucket@namespace/conda_environments/cpu/Data_
↳Exploration_and_Manipulation_for_CPU_Python_3.7/2.0/dataexpl_p37_cpu_v2"
```

(continues on next page)

(continued from previous page)

```

>>> inference_python_version = "3.7"
>>> if not os.path.exists(model_artifact_location):
...     os.makedirs(model_artifact_location)
>>> X, y = make_classification(n_samples=100, n_features=20, n_classes=2)
>>> lrmodel = LogisticRegression().fit(X, y)
>>> with open(os.path.join(model_artifact_location, 'model.pkl'), "wb") as mfile:
...     cloudpickle.dump(lrmodel, mfile)
>>> modelartifact = prepare_generic_model(
...     model_artifact_location,
...     model = lrmodel,
...     force_overwrite=True,
...     inference_conda_env=inference_conda_env,
...     ignore_deployment_error=True,
...     inference_python_version=inference_python_version
... )
>>> modelartifact.reload() # Call reload to update the ModelArtifact object with
↳ the generated score.py
>>> assert len(modelartifact.predict(X[:5])['prediction']) == 5 #Test the generated
↳ score.py works. This may require customization.
>>> with open(os.path.join(model_artifact_location, "runtime.yaml")) as rf:
...     content = yaml.load(rf, Loader=yaml.FullLoader)
...     assert content['MODEL_DEPLOYMENT']['INFERENCE_CONDA_ENV']['INFERENCE_ENV_
↳ PATH'] == inference_conda_env
...     assert content['MODEL_DEPLOYMENT']['INFERENCE_CONDA_ENV']['INFERENCE_PYTHON_
↳ VERSION'] == inference_python_version
>>> # Save Model to model artifact
>>> ocimodel = modelartifact.save(
...     project_id="ocil.....", # OCID of the project to which the model to be
↳ associated
...     compartment_id="ocil.....", # OCID of the compartment where the model will
↳ reside
...     display_name="LRModel_01",
...     description="My Logistic Regression Model",
...     ignore_pending_changes=True,
...     timeout=100,
...     ignore_introspection=True,
... )
>>> print(f"The OCID of the model is: {ocimodel.id}")

```

**Returns****model\_artifact** – A generic model artifact**Return type**

ads.model\_artifact.model\_artifact

ads.common.model\_export\_util.serialize\_model(model=None, target\_dir=None, X=None, y=None, model\_type=None, \*\*kwargs)

**Parameters**

- **model** (ads.Model) – A model to be serialized
- **target\_dir** (str, optional) – directory to output the serialized model

- **X** (*Union[pandas.DataFrame, pandas.Series]*) – The X data
- **y** (*Union[list, pandas.DataFrame, pandas.Series]*) – Tbe Y data
- **model\_type** (*str, optional*) – A string corresponding to the model type

**Returns**

**model\_kwargs** – A dictionary of model kwargs for the serialized model

**Return type**

Dict

**18.1.1.3.11 ads.common.function.fn\_util module**

`ads.common.function.fn_util.generate_fn_artifacts(path: str, fn_name: Optional[str] = None, fn_attributes=None, artifact_type_generic=False, **kwargs)`

Generates artifacts for fn (<https://fnproject.io>) at the provided path -

- func.py
- func.yaml
- requirements.txt if not there. If exists appends fdk to the file.
- score.py

**Parameters**

- **path** (*str*) – Target folder where the artifacts are placed.
- **fn\_attributes** (*dict*) – dictionary specifying all the function attributes as described in <https://github.com/fnproject/docs/blob/master/fn/develop/func-file.md>
- **artifact\_type\_generic** (*bool*) – default is False. This attribute decides which template to pick for score.py. If True, it is assumed that the code to load is provided by the user.

`ads.common.function.fn_util.get_function_config()` → dict

Returns dictionary loaded from func\_conf.yaml

`ads.common.function.fn_util.prepare_fn_attributes(func_name: str, schema_version=20180708, version=None, python_runtime=None, entry_point=None, memory=None)` → dict

Workaround for collections.namedtuples. The defaults are not supported.

`ads.common.function.fn_util.write_score(path, **kwargs)`

**18.1.1.3.12 ads.common.utils module**

**exception** `ads.common.utils.FileOverwriteError`

Bases: Exception

**class** `ads.common.utils.JsonConverter(*, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None)`

Bases: `JSONEncoder`

Constructor for `JSONEncoder`, with sensible defaults.

If `skipkeys` is `false`, then it is a `TypeError` to attempt encoding of keys that are not `str`, `int`, `float` or `None`. If `skipkeys` is `True`, such items are simply skipped.

If `ensure_ascii` is `true`, the output is guaranteed to be `str` objects with all incoming non-ASCII characters escaped. If `ensure_ascii` is `false`, the output can contain non-ASCII characters.

If `check_circular` is `true`, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an `OverflowError`). Otherwise, no such check takes place.

If `allow_nan` is `true`, then `NaN`, `Infinity`, and `-Infinity` will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a `ValueError` to encode such floats.

If `sort_keys` is `true`, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an (`item_separator`, `key_separator`) tuple. The default is `(',', ':')` if `indent` is `None` and `(' ', ':')` otherwise. To get the most compact JSON representation, you should specify `(',', ':')` to eliminate whitespace.

If specified, `default` is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

**default**(*obj*)

Converts an object to JSON based on its type

**Parameters**

**obj** (`Object`) – An object which is being converted to `Json`, supported types are `pandas` `Timestamp`, `series`, `dataframe`, or `categorical` or `numpy` `ndarrays`.

**Returns**

`Json`

**Return type**

A `json` representation of the object.

`ads.common.utils.batch_convert_case(spec: dict, to_fmt: str) → Dict`

Convert the case of a dictionary of `spec` from `camel` to `snake` or vice versa.

**Parameters**

- **spec** (`dict`) – dictionary of `spec` to convert
- **to\_fmt** (`str`) – format to convert to, can be “`camel`” or “`snake`”

**Returns**

dictionary of converted `spec`

**Return type**

`dict`

`ads.common.utils.camel_to_snake(name: str) → str`

Converts the `camel` case string to the `snake` representation.

**Parameters**

**name** (*str*) – The name to convert.

**Returns**

*str*

**Return type**

The name converted to the snake representation.

```
ads.common.utils.copy_file(uri_src: str, uri_dst: str, force_overwrite: Optional[bool] = False, auth:
    Optional[Dict] = None, chunk_size: Optional[int] = 8192,
    progressbar_description: Optional[str] = 'Copying {uri_src}` to `{uri_dst}`') →
    str
```

Copies file from *uri\_src* to *uri\_dst*. If *uri\_dst* specifies a directory, the file will be copied into *uri\_dst* using the base filename from *uri\_src*. Returns the path to the newly created file.

**Parameters**

- **uri\_src** (*str*) – The URI of the source file, which can be local path or OCI object storage URI.
- **uri\_dst** (*str*) – The URI of the destination file, which can be local path or OCI object storage URI.
- **force\_overwrite** ((*bool*, *optional*). Defaults to *False*.) – Whether to overwrite existing files or not.
- **auth** ((*Dict*, *optional*). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **chunk\_size** ((*int*, *optional*). Defaults to *DEFAULT\_BUFFER\_SIZE*) – How much data can be copied in one iteration.

**Returns**

The path to the newly created file.

**Return type**

*str*

**Raises**

**FileExistsError** – If a destination file exists and *force\_overwrite* set to *False*.

```
ads.common.utils.copy_from_uri(uri: str, to_path: str, unpack: Optional[bool] = False, force_overwrite:
    Optional[bool] = False, auth: Optional[Dict] = None) → None
```

Copies file(s) to local path. Can be a folder, archived folder or a separate file. The source files can be located in a local folder or in OCI Object Storage.

**Parameters**

- **uri** (*str*) – The URI of the source file or directory, which can be local path or OCI object storage URI.
- **to\_path** (*str*) – The local destination path. If this is a directory, the source files will be placed under it.
- **unpack** ((*bool*, *optional*). Defaults to *False*.) – Indicate if zip or tar.gz file specified by the uri should be unpacked. This option has no effect on other files.
- **force\_overwrite** ((*bool*, *optional*). Defaults to *False*.) – Whether to overwrite existing files or not.

- **auth** (*(Dict, optional). Defaults to None.*) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

**Returns**

Nothing

**Return type**

None

**Raises**

**ValueError** – If destination path is already exist and *force\_overwrite* is set to False.

`ads.common.utils.download_from_web(url: str, to_path: str) → None`

Downloads a single file from http/https/ftp.

**Parameters**

- **url** (*str*) – The URL of the source file.
- **to\_path** (*path-like object*) – Local destination path.

**Returns**

Nothing

**Return type**

None

`ads.common.utils.ellipsis_strings(raw, n=24)`

takes a sequence (<string>, list(<string>), tuple(<string>), pd.Series(<string>) and Ellipsis'ize them at position n

`ads.common.utils.extract_lib_dependencies_from_model(model) → dict`

Extract a dictionary of library dependencies for a model

**Parameters**

**model** –

**Returns**

**Dict**

**Return type**

A dictionary of library dependencies.

`ads.common.utils.first_not_none(itr)`

Returns the first non-none result from an iterable, similar to `any()` but return value not true/false

`ads.common.utils.flatten(d, parent_key="")`

Flattens nested dictionaries to a single layer dictionary

**Parameters**

- **d** (*dict*) – The dictionary that needs to be flattened
- **parent\_key** (*str*) – Keys in the dictionary that are nested

**Returns**

**a\_dict** – a single layer dictionary

**Return type**

dict



`ads.common.utils.folder_size(path: str) → int`

Recursively calculating a size of the *path* folder.

**Parameters**

**path** (*str*) – Path to the folder.

**Returns**

The size fo the folder in bytes.

**Return type**

int

`ads.common.utils.generate_requirement_file(requirements: dict, file_path: str, file_name: str = 'requirements.txt')`

Generate requirements file at *file\_path*.

**Parameters**

- **requirements** (*dict*) – Key is the library name and value is the version
- **file\_path** (*str*) – Directory to save requirements.txt
- **file\_name** (*str*) – Optional parameter to specify the file name

`ads.common.utils.get_base_modules(model)`

Get the base modules from an ADS model

`ads.common.utils.get_bootstrap_styles()`

Returns HTML bootstrap style information

`ads.common.utils.get_compute_accelerator_ncores()`

`ads.common.utils.get_cpu_count()`

Returns the number of CPUs available on this machine

`ads.common.utils.get_dataframe_styles(max_width=75)`

Styles used for dataframe, example usage:

```
df.style.set_table_styles(utils.get_dataframe_styles()).set_table_attributes('class=table').render()
```

**Returns**

**styles** – A list of dataframe table styler styles.

**Return type**

array

`ads.common.utils.get_files(directory: str)`

List out all the file names under this directory.

**Parameters**

**directory** (*str*) – The directory to list out all the files from.

**Returns**

List of the files in the directory.

**Return type**

List

`ads.common.utils.get_oci_config()`

Returns the OCI config location, and the OCI config profile.

`ads.common.utils.get_progress_bar(max_progress, description='Initializing')`

this will return an instance of `ProgressBar`, sensitive to the runtime environment

`ads.common.utils.get_random_name_for_resource()` → str

Returns randomly generated easy to remember name. It consists from 1 adjective and 1 animal word, tailed by UTC timestamp (joined with '-'). This is an ADS default resource name generated for models, jobs, jobruns, model deployments, pipelines.

**Returns**

Randomly generated easy to remember name for oci resources - models, jobs, jobruns, model deployments, pipelines. Example: polite-panther-2022-08-17-21:15.46; strange-spider-2022-08-17-23:55.02

**Return type**

str

`ads.common.utils.get_sqlalchemy_engine(connection_url, *args, **kwargs)`

The SQLAlchemy docs say to use a single engine per connection\_url, this class will take care of that.

**Parameters**

**connection\_url** (*string*) – The URL to connect to

**Returns**

**engine** – The engine from which SQLAlchemy commands can be ran on

**Return type**

SQLAlchemy engine

`ads.common.utils.get_value(obj, attr, default=None)`

Gets a copy of the value from a nested dictionary of an object with nested attributes.

**Parameters**

- **obj** – An object or a dictionary
- **attr** – Attributes as a string separated by dot(.)
- **default** – Default value to be returned if attribute is not found.

**Returns**

A copy of the attribute value. For dict or list, a deepcopy will be returned.

**Return type**

Any

`ads.common.utils.highlight_text(text)`

Returns text with html highlights. :param text: The text to be highlighted. :type text: String

**Returns**

**ht** – The text with html highlight information.

**Return type**

*String*

`ads.common.utils.horizontal_scrollable_div(html)`

Wrap html with the necessary html to make horizontal scrolling possible.

## Examples

```
display(HTML(utils.horizontal_scrollable_div(my_html)))
```

### Parameters

**html** (*str*) – Your HTML to wrap.

### Returns

Wrapped HTML.

### Return type

type

```
ads.common.utils.human_size(num_bytes: int, precision: Optional[int] = 2) → str
```

Converts bytes size to a string representing its value in B, KB, MB and GB.

### Parameters

- **num\_bytes** (*int*) – The size in bytes.
- **precision** (*((int, optional). Defaults to 2.)*) – The precision of converting the bytes value.

### Returns

A string representing the size in B, KB, MB and GB.

### Return type

str

```
ads.common.utils.inject_and_copy_kwargs(kwargs, **args)
```

Takes in a dictionary and returns a copy with the args injected

## Examples

```
>>> foo(arg1, args, utils.inject_and_copy_kwargs(kwargs, arg3=12, arg4=42))
```

### Parameters

- **kwargs** (*dict*) – The original *kwargs*.
- **\*\*args** (*type*) – A series of arguments, foo=42, bar=12 etc

### Returns

**d** – new dictionary object that you can use in place of *kwargs*

### Return type

dict

```
ads.common.utils.is_data_too_wide(data: Union[list, tuple, Series, ndarray, DataFrame], max_col_num: int) → bool
```

Returns true if the data has too many columns.

### Parameters

- **data** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]*) – A sample of data that will be used to generate schema.
- **max\_col\_num** (*int.*) – The maximum column size of the data that allows to auto generate schema.

`ads.common.utils.is_debug_mode()`

Returns true if ADS is in debug mode.

`ads.common.utils.is_documentation_mode()`

Returns true if ADS is in documentation mode.

`ads.common.utils.is_notebook()`

Returns true if the environment is a jupyter notebook.

`ads.common.utils.is_resource_principal_mode()`

Returns true if ADS is in resource principal mode.

`ads.common.utils.is_same_class(obj, cls)`

checks to see if object is the same class as cls

`ads.common.utils.is_test()`

Returns true if ADS is in test mode.

**class** `ads.common.utils.ml_task_types(value)`

Bases: Enum

An enumeration.

**BINARY\_CLASSIFICATION** = 2

**BINARY\_TEXT\_CLASSIFICATION** = 4

**MULTI\_CLASS\_CLASSIFICATION** = 3

**MULTI\_CLASS\_TEXT\_CLASSIFICATION** = 5

**REGRESSION** = 1

**UNSUPPORTED** = 6

`ads.common.utils.numeric_pandas_dtypes()`

Returns a list of the “numeric” pandas data types

`ads.common.utils.oci_config_file()`

Returns the OCI config file location

`ads.common.utils.oci_config_location()`

Returns oci configuration file location.

`ads.common.utils.oci_config_profile()`

Returns the OCI config profile location.

`ads.common.utils.oci_key_location()`

Returns the OCI key location

`ads.common.utils.oci_key_profile()`

Returns key profile value specified in oci configuration file.

`ads.common.utils.print_user_message(msg, display_type='tip', see_also_links=None, title='Tip')`

This method is deprecated and will be removed in future releases. Prints in html formatted block one of tip|info|warn type.

#### Parameters

- **msg** (*str* or *list*) – The actual message to display. `display_type` is “module”, `msg` can be a list of [module name, module package name], i.e. [“automl”, “ads[ml]”]

- **display\_type** (*str* (default 'tip')) – The type of user message.
- **see\_also\_links** (*list of tuples in the form of [('display\_name', 'url')]*) –
- **title** (*str* (default 'tip')) – The title of user message.

`ads.common.utils.random_valid_ocid(prefix='ocidl.dataflowapplication.oc1.iad')`

Generates a random valid ocid.

#### Parameters

**prefix** (*str*) – A prefix, corresponding to a region location.

#### Returns

**ocid** – a valid ocid with the given prefix.

#### Return type

*str*

`ads.common.utils.remove_file(file_path: str, auth: Optional[Dict] = None) → None`

Reoves file.

#### Parameters

- **file\_path** (*str*) – The path of the source file, which can be local path or OCI object storage URI.
- **auth** (*(Dict, optional). Defaults to None.*) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

#### Returns

Nothing.

#### Return type

None

`ads.common.utils.replace_spaces(lst)`

Replace all spaces with underscores for strings in the list.

Requires that the list contains strings for each element.

lst: list of strings

`ads.common.utils.set_oci_config(oci_config_location, oci_config_profile)`

#### Parameters

- **oci\_config\_location** – location of the config file, for example, `~/oci/config`
- **oci\_config\_profile** – The profile to load from the config file. Defaults to “DEFAULT”

`ads.common.utils.snake_to_camel(name: str, capitalized_first_token: Optional[bool] = False) → str`

Converts the snake case string to the camel representation.

#### Parameters

- **name** (*str*) – The name to convert.
- **capitalized\_first\_token** (*(bool, optional). Defaults to False.*) – Wether the first token needs to be capitalized or not.

#### Returns

*str*

**Return type**

The name converted to the camel representation.

`ads.common.utils.split_data(X, y, random_state=42, test_size=0.3)`

Splits data using Sklearn based on the input type of the data.

**Parameters**

- **X** (a *Pandas Dataframe*) – The data points.
- **y** (a *Pandas Dataframe*) – The labels.
- **random\_state** (*int*) – A random state for reproducibility.
- **test\_size** (*int*) – The number of elements that should be included in the test dataset.

`ads.common.utils.to_dataframe(data: Union[list, tuple, Series, ndarray, DataFrame])`

Convert to pandas DataFrame.

**Parameters**

**data** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]*) – Convert data to pandas DataFrame.

**Returns**

pandas DataFrame.

**Return type**

pd.DataFrame

`ads.common.utils.truncate_series_top_n(series, n=24)`

take a series which can be interpreted as a dict, index=key, this function sorts by the values and takes the top-n values, and returns a new series

`ads.common.utils.wrap_lines(li, heading="")`

Wraps the elements of iterable into multi line string of fixed width

### 18.1.1.3.13 Module contents

#### 18.1.1.3.14 ads.common.model\_metadata\_mixin module

**class** `ads.common.model_metadata_mixin.MetadataMixin`

Bases: object

MetadataMixin class which populates the custom metadata, taxonomy metadata, input/output schema and provenance metadata.

**populate\_metadata**(*use\_case\_type: Optional[str] = None, data\_sample: Optional[ADSDData] = None, X\_sample: Optional[Union[list, tuple, DataFrame, Series, ndarray]] = None, y\_sample: Optional[Union[list, tuple, DataFrame, Series, ndarray]] = None, training\_script\_path: Optional[str] = None, training\_id: Optional[str] = None, ignore\_pending\_changes: bool = True, max\_col\_num: int = 2000*)

Populates input schema and output schema. If the schema exceeds the limit of 32kb, save as json files to the artifact directory.

**Parameters**

- **use\_case\_type** ((*str, optional*). Defaults to None.) – The use case type of the model.

- **data\_sample** (*ADSDData, optional*). Defaults to *None*.) – A sample of the data that will be used to generate input\_schema and output\_schema.
- **X\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]*. Defaults to *None*.) – A sample of input data that will be used to generate input schema.
- **y\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]*. Defaults to *None*.) – A sample of output data that will be used to generate output schema.
- **training\_script\_path** (*str*. Defaults to *None*.) – Training script path.
- **training\_id** (*(str, optional)*. Defaults to *None*.) – The training model OCID.
- **ignore\_pending\_changes** (*bool*. Defaults to *False*.) – Ignore the pending changes in git.
- **max\_col\_num** (*((int, optional)*. Defaults to *utils.DATA\_SCHEMA\_MAX\_COL\_NUM*.) – The maximum number of columns allowed in auto generated schema.

**Returns**

Nothing.

**Return type**

None

**populate\_schema**(*data\_sample: Optional[ADSDData] = None, X\_sample: Optional[Union[List, Tuple, DataFrame, Series, ndarray]] = None, y\_sample: Optional[Union[List, Tuple, DataFrame, Series, ndarray]] = None, max\_col\_num: int = 2000*)

Populate input and output schemas. If the schema exceeds the limit of 32kb, save as json files to the artifact dir.

**Parameters**

- **data\_sample** (*ADSDData*) – A sample of the data that will be used to generate input\_schema and output\_schema.
- **X\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]*) – A sample of input data that will be used to generate the input schema.
- **y\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]*) – A sample of output data that will be used to generate the output schema.
- **max\_col\_num** (*((int, optional)*. Defaults to *utils.DATA\_SCHEMA\_MAX\_COL\_NUM*.) – The maximum number of columns allowed in auto generated schema.

**18.1.1.4 ads.bds package****18.1.1.4.1 Submodules****18.1.1.4.2 ads.bds.auth module****exception ads.bds.auth.KRB5KinitError**

Bases: Exception

KRB5KinitError class when kinit -kt command failed to generate cached ticket with the keytab file and the krb5 config file.

`ads.bds.auth.has_kerberos_ticket()`

Whether kerberos cache ticket exists.

`ads.bds.auth.init_ccache_with_keytab(principal: str, keytab_file: str) → None`

Initialize credential cache using keytab file.

**Parameters**

- **principal** (*str*) – The unique identity to which Kerberos can assign tickets.
- **keytab\_path** (*str*) – Path to your keytab file.

**Returns**

Nothing.

**Return type**

None

`ads.bds.auth.krbcontext(principal: str, keytab_path: str, krb5_path: str = '~/.bds_config/krb5.conf') → None`

A context manager for Kerberos-related actions. It provides a Kerberos context that you can put code inside. It will initialize credential cache automatically with keytab if no cached ticket exists. Otherwise, does nothing.

**Parameters**

- **principal** (*str*) – The unique identity to which Kerberos can assign tickets.
- **keytab\_path** (*str*) – Path to your keytab file.
- **krb5\_path** ((*str*, *optional*)) – Path to your krb5 config file.

**Returns**

Nothing.

**Return type**

None

## Examples

```
>>> from ads.bds.auth import krbcontext
>>> from pyhive import hive
>>> with krbcontext(principal = "your_principal", keytab_path = "your_keytab_path"):
>>>     hive_cursor = hive.connect(host="your_hive_host",
...                               port="your_hive_port",
...                               auth='KERBEROS',
...                               kerberos_service_name="hive").cursor()
```

`ads.bds.auth.refresh_ticket(principal: str, keytab_path: str, krb5_path: str = '~/.bds_config/krb5.conf') → None`

generate new cached ticket based on the principal and keytab file path.

**Parameters**

- **principal** (*str*) – The unique identity to which Kerberos can assign tickets.
- **keytab\_path** (*str*) – Path to your keytab file.
- **krb5\_path** ((*str*, *optional*)) – Path to your krb5 config file.

**Returns**

Nothing.



**Return type**

None

**Examples**

```
>>> from ads.bds.auth import refresh_ticket
>>> from pyhive import hive
>>> refresh_ticket(principal = "your_principal", keytab_path = "your_keytab_path")
>>> hive_cursor = hive.connect(host="your_hive_host",
...                             port="your_hive_port",
...                             auth='KERBEROS',
...                             kerberos_service_name="hive").cursor()
```

**18.1.1.4.3 Module contents****18.1.1.5 ads.data\_labeling package****18.1.1.5.1 Submodules****18.1.1.5.2 ads.data\_labeling.interface.loader module**

**class** ads.data\_labeling.interface.loader.Loader

Bases: ABC

Data Loader Interface.

**abstract load**(\*\*kwargs) → Any

**18.1.1.5.3 ads.data\_labeling.interface.parser module**

**class** ads.data\_labeling.interface.parser.Parser

Bases: ABC

Data Parser Interface.

**abstract parse**() → Any

**18.1.1.5.4 ads.data\_labeling.interface.reader module**

**class** ads.data\_labeling.interface.reader.Reader

Bases: ABC

Data Reader Interface.

**info**() → Serializable

**abstract read**() → Any

#### 18.1.1.5.5 ads.data\_labeling.boundingbox module

```
class ads.data_labeling.boundingbox.BoundingBoxItem(top_left: ~typing.Tuple[float, float], bottom_left:  
~typing.Tuple[float, float], bottom_right:  
~typing.Tuple[float, float], top_right:  
~typing.Tuple[float, float], labels:  
~typing.List[str] = <factory>)
```

Bases: object

BoundingBoxItem class representing bounding box label.

##### **labels**

List of labels for this bounding box.

##### **Type**

List[str]

##### **top\_left**

Top left corner of this bounding box.

##### **Type**

Tuple[float, float]

##### **bottom\_left**

Bottom left corner of this bounding box.

##### **Type**

Tuple[float, float]

##### **bottom\_right**

Bottom right corner of this bounding box.

##### **Type**

Tuple[float, float]

##### **top\_right**

Top right corner of this bounding box.

##### **Type**

Tuple[float, float]

#### **Examples**

```
>>> item = BoundingBoxItem(  
...     labels = ['cat', 'dog'],  
...     bottom_left=(0.2, 0.4),  
...     top_left=(0.2, 0.2),  
...     top_right=(0.8, 0.2),  
...     bottom_right=(0.8, 0.4))  
>>> item.to_yolo(categories = ['cat', 'dog', 'horse'])
```

**bottom\_left**: Tuple[float, float]

**bottom\_right**: Tuple[float, float]

**classmethod** `from_yolo(bbox: List[Tuple], categories: Optional[List[str]] = None) → BoundingBoxItem`

Converts the YOLO formatted annotations to BoundingBoxItem.

**Parameters**

- **bboxes** (*List[Tuple]*) – The list of bounding box annotations in YOLO format. Example: [(0, 0.511560675, 0.50234826, 0.47013485, 0.57803468)]
- **categories** (*List[str]*) – The list of object categories in proper order for model training. Example: ['cat','dog','horse']

**Returns**

The BoundingBoxItem.

**Return type**

*BoundingBoxItem*

**Raises**

**TypeError** – When categories list has a wrong format.

**labels: List[str]**

**to\_yolo(categories: List[str]) → List[Tuple[int, float, float, float, float]]**

Converts BoundingBoxItem to the YOLO format.

**Parameters**

**categories** (*List[str]*) – The list of object categories in proper order for model training. Example: ['cat','dog','horse']

**Returns**

The list of YOLO formatted bounding boxes.

**Return type**

List[Tuple[int, float, float, float, float]]

**Raises**

- **ValueError** – When categories list not provided. When categories list not matched with the labels.
- **TypeError** – When categories list has a wrong format.

**top\_left: Tuple[float, float]**

**top\_right: Tuple[float, float]**

**class** `ads.data_labeling.boundingbox.BoundingBoxItems(items: ~typing.List[~ads.data_labeling.boundingbox.BoundingBoxItem] = <factory>)`

Bases: object

BoundingBoxItems class which consists of a list of BoundingBoxItem.

**items**

List of BoundingBoxItem.

**Type**

List[*BoundingBoxItem*]

## Examples

```
>>> item = BoundingBoxItem(
...     labels = ['cat', 'dog']
...     bottom_left=(0.2, 0.4),
...     top_left=(0.2, 0.2),
...     top_right=(0.8, 0.2),
...     bottom_right=(0.8, 0.4))
>>> items = BoundingBoxItems(items = [item])
>>> items.to_yolo(categories = ['cat', 'dog', 'horse'])
```

**items:** List[*BoundingBoxItem*]

**to\_yolo**(categories: List[str]) → List[Tuple[int, float, float, float, float]]

Converts BoundingBoxItems to the YOLO format.

### Parameters

**categories** (List[str]) – The list of object categories in proper order for model training.  
Example: ['cat', 'dog', 'horse']

### Returns

The list of YOLO formatted bounding boxes.

### Return type

List[Tuple[int, float, float, float, float]]

### Raises

- **ValueError** – When categories list not provided. When categories list not matched with the labels.
- **TypeError** – When categories list has a wrong format.

### 18.1.1.5.6 ads.data\_labeling.constants module

**class** ads.data\_labeling.constants.AnnotationType

Bases: object

AnnotationType class which contains all the annotation types that data labeling service supports.

**BOUNDING\_BOX** = 'BOUNDING\_BOX'

**ENTITY\_EXTRACTION** = 'ENTITY\_EXTRACTION'

**MULTI\_LABEL** = 'MULTI\_LABEL'

**SINGLE\_LABEL** = 'SINGLE\_LABEL'

**class** ads.data\_labeling.constants.DatasetType

Bases: object

DatasetType class which contains all the dataset types that data labeling service supports.

**DOCUMENT** = 'DOCUMENT'

**IMAGE** = 'IMAGE'

**TEXT** = 'TEXT'

```
class ads.data_labeling.constants.Formats
```

Bases: object

Common formats class which contains all the common formats that are supported to convert to.

```
SPACY = 'spacy'
```

```
YOLO = 'yolo'
```

#### 18.1.1.5.7 ads.data\_labeling.data\_labeling\_service module

```
class ads.data_labeling.data_labeling_service.DataLabeling(compartment_id: Optional[str] =
    None, dls_cp_client_auth:
    Optional[dict] = None,
    dls_dp_client_auth: Optional[dict] =
    None)
```

Bases: OCIWorkRequestMixin

Class for data labeling service. Integrate the data labeling service APIs.

#### Examples

```
>>> import ads
>>> import pandas
>>> from ads.data_labeling.data_labeling_service import DataLabeling
>>> ads.set_auth("api_key")
>>> dls = DataLabeling()
>>> dls.list_dataset()
>>> metadata_path = dls.export(dataset_id="your dataset id",
...     path="oci://<bucket_name>@<namespace>/folder")
>>> df = pd.DataFrame.ads.read_labeled_data(metadata_path)
```

Initialize a DataLabeling class.

#### Parameters

- **compartment\_id** (*str, optional*) – OCID of data labeling datasets' compartment
- **dls\_cp\_client\_auth** (*dict, optional*) – Data Labeling control plane client auth. Default is None. The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **dls\_dp\_client\_auth** (*dict, optional*) – Data Labeling data plane client auth. Default is None. The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

#### Returns

Nothing.

#### Return type

None

**export**(*dataset\_id: str, path: str, include\_unlabeled=False*) → str

Export dataset based on the *dataset\_id* and save the jsonl files under the path (metadata jsonl file and the records jsonl file) to the object storage path provided by the user and return the metadata jsonl path.

**Parameters**

- **dataset\_id** (*str*) – The dataset id of which the snapshot will be generated.
- **path** (*str*) – The object storage path to store the generated snapshot.  
“oci://<bucket\_name>@<namespace>/prefix”
- **include\_unlabeled** (*bool, Optional. Defaults to False.*) – Whether to include unlabeled records or not.

**Returns**

oci path of the metadata jsonl file.

**Return type**

str

**list\_dataset**(*\*\*kwargs*) → DataFrame

List all the datasets created from the data labeling service under a given compartment.

**Parameters**

**kwargs** (*dict, optional*) – Additional keyword arguments will be passed to `oci.data_labeling_serviceDataLabelingManagementClient.list_datasets` method.

**Returns**

pandas dataframe which contains the dataset information.

**Return type**

pandas.DataFrame

**Raises**

**Exception** – If `pagination.list_call_get_all_results()` fails

#### 18.1.1.5.8 ads.data\_labeling.metadata module

```
class ads.data_labeling.metadata.Metadata(source_path: str = "", records_path: str = "", labels:
    ~typing.List[str] = <factory>, dataset_name: str = "",
    compartment_id: str = "", dataset_id: str = "",
    annotation_type: str = "", dataset_type: str = "")
```

Bases: `DataClassSerializable`

The class that representing the labeled dataset metadata.

**source\_path**

Contains information on where all the source data(image/text/document) stores.

**Type**

str

**records\_path**

Contains information on where records jsonl file stores.

**Type**

str

**labels**

List of classes/labels for the dataset.

**Type**

List

**dataset\_name**

Dataset display name on the Data Labeling Service console.

**Type**

str

**compartment\_id**

Compartment id of the labeled dataset.

**Type**

str

**dataset\_id**

Dataset id.

**Type**

str

**annotation\_type**

Type of the labeling/annotation task. Currently supports SINGLE\_LABEL, MULTI\_LABEL, ENTITY\_EXTRACTION, BOUNDING\_BOX.

**Type**

str

**dataset\_type**

Type of the dataset. Currently supports Text, Image, DOCUMENT.

**Type**

str

```
annotation_type: str = ''
```

```
compartment_id: str = ''
```

```
dataset_id: str = ''
```

```
dataset_name: str = ''
```

```
dataset_type: str = ''
```

```
classmethod from_dls_dataset(dataset: Dataset) → Metadata
```

Constructs a Metadata instance from OCI DLS dataset.

**Parameters**

**dataset** (*OCIDLSDataset*) – OCIDLSDataset object.

**Returns**

The ads labeled dataset metadata instance.

**Return type**

*Metadata*

```
labels: List[str]
```

**records\_path:** str = ''

**source\_path:** str = ''

**to\_dataframe()** → DataFrame

Converts the metadata to dataframe format.

**Returns**

The metadata in Pandas dataframe format.

**Return type**

pandas.DataFrame

**to\_dict()** → Dict

Converts to dictionary representation.

**Returns**

The metadata in dictionary type.

**Return type**

Dict

#### 18.1.1.5.9 ads.data\_labeling.ner module

**class** ads.data\_labeling.ner.NERItem(*label: str = "", offset: int = 0, length: int = 0*)

Bases: object

NERItem class which is a representation of a token span.

**label**

Entity name.

**Type**

str

**offset**

The token span's entity start index position in the text.

**Type**

int

**length**

Length of the token span.

**Type**

int

**classmethod** from\_spacy(*token*) → *NERItem*

**label:** str = ''

**length:** int = 0

**offset:** int = 0

**to\_spacy()** → tuple

Converts one NERItem to the spacy format.

**Returns**

NERItem in the spacy format



**Return type**

Tuple

**class** `ads.data_labeling.ner.NERItems`(*items*: *~typing.List[~ads.data\_labeling.ner.NERItem]* = <factory>)

Bases: `object`

NERItems class consists of a list of `NERItem`.

**items**

List of `NERItem`.

**Type**List[[NERItem](#)]

**items**: List[[NERItem](#)]

**to\_spacy()** → List[tuple]

Converts `NERItems` to the spacy format.

**Returns**

List of `NERItems` in the Spacy format.

**Return type**

List[tuple]

**exception** `ads.data_labeling.ner.WrongEntityFormatLabelIsEmpty`

Bases: `ValueError`

**exception** `ads.data_labeling.ner.WrongEntityFormatLabelNotString`

Bases: `ValueError`

**exception** `ads.data_labeling.ner.WrongEntityFormatLengthIsNegative`

Bases: `ValueError`

**exception** `ads.data_labeling.ner.WrongEntityFormatLengthNotInteger`

Bases: `ValueError`

**exception** `ads.data_labeling.ner.WrongEntityFormatOffsetIsNegative`

Bases: `ValueError`

**exception** `ads.data_labeling.ner.WrongEntityFormatOffsetNotInteger`

Bases: `ValueError`

**18.1.1.5.10 ads.data\_labeling.record module**

**class** `ads.data_labeling.record.Record`(*path*: *str* = "", *content*: *Optional[Any]* = *None*, *annotation*: *Optional[Union[Tuple, str, List[[BoundingBoxItem](#)], List[[NERItem](#)]]]* = *None*)

Bases: `object`

Class representing `Record`.

**path**

File path.

**Type**

str

**content**

Content of the record.

**Type**

Any

**annotation**

Annotation/label of the record.

**Type**

Union[Tuple, str, List[*BoundingBoxItem*], List[*NERItem*]]

**annotation:** Union[Tuple, str, List[*BoundingBoxItem*], List[*NERItem*]] = None

**content:** Any = None

**path:** str = ''

**to\_dict()** → Dict

Convert the Record instance to a dictionary.

**Returns**

Dictionary representation of the Record instance.

**Return type**

Dict

**to\_tuple()** → Tuple[str, Any, Union[Tuple, str, List[*BoundingBoxItem*], List[*NERItem*]]]

Convert the Record instance to a tuple.

**Returns**

Tuple representation of the Record instance.

**Return type**

Tuple

#### 18.1.1.5.11 *ads.data\_labeling.mixin.data\_labeling* module

**class** *ads.data\_labeling.mixin.data\_labeling.DataLabelingAccessMixin*

Bases: object

Mixin class for labeled text data.

**static read\_labeled\_data**(*path: Optional[str] = None, dataset\_id: Optional[str] = None, compartment\_id: Optional[str] = None, auth: Optional[Dict] = None, materialize: bool = False, encoding: str = 'utf-8', include\_unlabeled: bool = False, format: Optional[str] = None, chunksize: Optional[int] = None*)

Loads the dataset generated by data labeling service from either the export file or the Data Labeling Service.

**Parameters**

- **path** ((*str, optional*). Defaults to None) – The export file path, can be either local or object storage path.
- **dataset\_id** ((*str, optional*). Defaults to None) – The dataset OCID.
- **compartment\_id** (*str*. Defaults to the *compartment\_id* from the *env variable*.) – The compartment OCID of the dataset.

- **auth**((dict, optional). Defaults to None) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **materialize**((bool, optional). Defaults to False) – Whether the content of the dataset file should be loaded or it should return the file path to the content. By default the content will not be loaded.
- **encoding**((str, optional). Defaults to 'utf-8') – Encoding of files. Only used for “TEXT” dataset.
- **include\_unlabeled**((bool, optional). Default to False) – Whether to load the unlabeled records or not.
- **format**((str, optional). Defaults to None) – Output format of annotations. Can be None, “spacy” for dataset Entity Extraction type or “yolo” for Object Detection type.
  - When None, it outputs List[NERItem] or List[BoundingBoxItem],
  - When “spacy”, it outputs List[Tuple],
  - When “yolo”, it outputs List[List[Tuple]].
- **chunksize**((int, optional). Defaults to None) – The amount of records that should be read in one iteration. The result will be returned in a generator format.

#### Returns

`pd.DataFrame` if `chunksize` is not specified. `Generator[pd.DataFrame]` if `chunksize` is specified.

#### Return type

`Union[Generator[pd.DataFrame, Any, Any], pd.DataFrame]`

### Examples

```
>>> import pandas as pd
>>> import ads
>>> from ads.common import auth as authutil
>>> df = pd.DataFrame.ads.read_labeled_data(path="path_to_your_metadata.jsonl",
...                                     auth=authutil.api_keys(),
...                                     materialize=False)
```

	Path	Content	Annotations
0	path/to/the/content/file		yes
1	path/to/the/content/file		no

```
>>> df = pd.DataFrame.ads.read_labeled_data_from_dls(dataset_id="your_dataset_
↵ocid",
...                                     compartment_id="your_
↵compartment_id",
...                                     auth=authutil.api_keys(),
...                                     materialize=False)
```

	Path	Content	Annotations
0	path/to/the/content/file		yes
1	path/to/the/content/file		no

**render\_bounding\_box**(*options: Optional[Dict] = None, content\_column: str = 'Content', annotations\_column: str = 'Annotations', categories: Optional[List[str]] = None, limit: int = 50, path: Optional[str] = None*) → None

Renders bounding box dataset. Displays only first 50 rows.

#### Parameters

- **options** (*dict*) – The colors options specified for rendering.
- **content\_column** (*Optional[str]*) – The column name with the content data.
- **annotations\_column** (*Optional[str]*) – The column name for the annotations list.
- **categories** (*Optional List[str]*) – The list of object categories in proper order for model training. Only used when bounding box annotations are in YOLO format. Example: ['cat','dog','horse']
- **limit** (*Optional[int]*. Defaults to 50) – The maximum amount of records to display.
- **path** (*Optional[str]*) – Path to save the image with annotations to local directory.

#### Returns

Nothing

#### Return type

None

#### Examples

```
>>> import pandas as pd
>>> import ads
>>> from ads.common import auth as authutil
>>> df = pd.DataFrame.ads.read_labeled_data(path="path_to_your_metadata.jsonl",
...                                       auth=authutil.api_keys(),
...                                       materialize=True)
>>> df.ads.render_bounding_box(content_column="Content", annotations_column=
↳ "Annotations")
```

**render\_ner**(*options: Dict = None, content\_column: str = 'Content', annotations\_column: str = 'Annotations', limit: int = 50*) → None

Renders NER dataset. Displays only first 50 rows.

#### Parameters

- **options** (*dict*) – The colors options specified for rendering.
- **content\_column** (*Optional[str]*) – The column name with the content data.
- **annotations\_column** (*Optional[str]*) – The column name for the annotations list.
- **limit** (*Optional[int]*. Defaults to 50) – The maximum amount of records to display.

#### Returns

Nothing

#### Return type

None

## Examples

```
>>> import pandas as pd
>>> import ads
>>> from ads.common import auth as authutil
>>> df = pd.DataFrame.ads.read_labeled_data(path="path_to_your_metadata.jsonl",
...                                       auth=authutil.api_keys(),
...                                       materialize=True)
>>> df.ads.render_ner(content_column="Content", annotations_column="Annotations
↪")
```

### 18.1.1.5.12 ads.data\_labeling.parser.export\_metadata\_parser module

**class** ads.data\_labeling.parser.export\_metadata\_parser.**MetadataParser**

Bases: *Parser*

MetadataParser class which parses the metadata from the record.

**EXPECTED\_KEYS** = ['id', 'compartmentId', 'displayName', 'labelsSet', 'annotationFormat', 'datasetSourceDetails', 'datasetFormatDetails']

**static parse**(*json\_data: Dict[Any, Any]*) → *Metadata*

Parses the metadata jsonl file.

**Parameters**

**json\_data** (*dict*) – dictionary format of the metadata jsonl file content.

**Returns**

Metadata object which contains the useful fields from the metadata jsonl file

**Return type**

*Metadata*

### 18.1.1.5.13 ads.data\_labeling.parser.export\_record\_parser module

**class** ads.data\_labeling.parser.export\_record\_parser.**BoundingBoxRecordParser**(*dataset\_source\_path: str, format: Optional[str] = None, categories: Optional[List[str]] = None*)

Bases: *RecordParser*

BoundingBoxRecordParser class which parses the label of BoundingBox label data.

Initiates a RecordParser instance.

**Parameters**

- **dataset\_source\_path** (*str*) – Dataset source path.
- **format** ((*str*, *optional*). Defaults to *None*.) – Output format of annotations.
- **categories** ((*List[str]*, *optional*). Defaults to *None*.) – The list of object categories in proper order for model training. Example: ['cat', 'dog', 'horse']

**Returns**

RecordParser instance.

**Return type**

*RecordParser*

```
class ads.data_labeling.parser.export_record_parser.EntityType
```

Bases: object

Entity type class for supporting multiple types of entities.

```
GENERIC = 'GENERIC'
```

```
IMAGEOBJECTSELECTION = 'IMAGEOBJECTSELECTION'
```

```
TEXTSELECTION = 'TEXTSELECTION'
```

```
class ads.data_labeling.parser.export_record_parser.MultiLabelRecordParser(dataset_source_path:  
                                                                           str, format:  
                                                                           Optional[str] =  
                                                                           None, categories:  
                                                                           Op-  
                                                                           tional[List[str]] =  
                                                                           None)
```

Bases: *RecordParser*

MultiLabelRecordParser class which parses the label of Multiple label data.

Initiates a RecordParser instance.

**Parameters**

- **dataset\_source\_path** (*str*) – Dataset source path.
- **format** ((*str, optional*). Defaults to *None*.) – Output format of annotations.
- **categories** ((*List[str], optional*). Defaults to *None*.) – The list of object categories in proper order for model training. Example: ['cat','dog','horse']

**Returns**

RecordParser instance.

**Return type**

*RecordParser*

```
class ads.data_labeling.parser.export_record_parser.NERRecordParser(dataset_source_path: str,  
                                                                      format: Optional[str] =  
                                                                      None, categories:  
                                                                      Optional[List[str]] =  
                                                                      None)
```

Bases: *RecordParser*

NERRecordParser class which parses the label of NER label data.

Initiates a RecordParser instance.

**Parameters**

- **dataset\_source\_path** (*str*) – Dataset source path.
- **format** ((*str, optional*). Defaults to *None*.) – Output format of annotations.

- **categories** (*List[str], optional*). Defaults to *None*.) – The list of object categories in proper order for model training. Example: ['cat','dog','horse']

**Returns**

RecordParser instance.

**Return type**

*RecordParser*

```
class ads.data_labeling.parser.export_record_parser.RecordParser(dataset_source_path: str,
                                                                format: Optional[str] = None,
                                                                categories: Optional[List[str]]
                                                                = None)
```

Bases: *Parser*

RecordParser class which parses the labels from the record.

**Examples**

```
>>> from ads.data_labeling.parser.export_record_parser import \
↳ SingleLabelRecordParser
>>> from ads.data_labeling.parser.export_record_parser import MultiLabelRecordParser
>>> from ads.data_labeling.parser.export_record_parser import NERRecordParser
>>> from ads.data_labeling.parser.export_record_parser import \
↳ BoundingBoxRecordParser
>>> import fsspec
>>> import json
>>> from ads.common import auth as authutil
>>> labels = []
>>> with fsspec.open("/path/to/records_file.jsonl", **authutil.api_keys()) as f:
>>>     for line in f:
>>>         bounding_box_labels = BoundingBoxRecordParser("source_data_path").
↳ parse(json.loads(line))
>>>         labels.append(bounding_box_labels)
```

Initiates a RecordParser instance.

**Parameters**

- **dataset\_source\_path** (*str*) – Dataset source path.
- **format** (*(str, optional)*). Defaults to *None*.) – Output format of annotations.
- **categories** (*List[str], optional*). Defaults to *None*.) – The list of object categories in proper order for model training. Example: ['cat','dog','horse']

**Returns**

RecordParser instance.

**Return type**

*RecordParser*

**parse**(*record: Dict*) → *Record*

Extracts the annotations from the record content. Constructs and returns a Record instance containing the file path and the labels.

**Parameters**

**record** (*Dict*) – Content of the record from the record file.

**Returns**

Record instance which contains the file path as well as the annotations.

**Return type**

*Record*

**class** `ads.data_labeling.parser.export_record_parser.RecordParserFactory`

Bases: `object`

`RecordParserFactory` class which contains a list of registered parsers and allows to register new `RecordParsers`.

**Current parsers include:**

- `SingleLabelRecordParser`
- `MultiLabelRecordParser`
- `NERRecordParser`
- `BoundingBoxRecordParser`

**static parser**(*annotation\_type: str, dataset\_source\_path: str, format: Optional[str] = None, categories: Optional[List[str]] = None*) → *RecordParser*

Gets the parser based on the `annotation_type`.

**Parameters**

- **annotation\_type** (*str*) – Annotation type which can be `SINGLE_LABEL`, `MULTI_LABEL`, `ENTITY_EXTRACTION` and `BOUNDING_BOX`.
- **dataset\_source\_path** (*str*) – Dataset source path.
- **format** ((*str, optional*). Defaults to *None*.) – Output format of annotations. Can be *None*, “spacy” for dataset Entity Extraction type or “yolo” for Object Detection type. When *None*, it outputs `List[NERItem]` or `List[BoundingBoxItem]`. When “spacy”, it outputs `List[Tuple]`. When “yolo”, it outputs `List[List[Tuple]]`.
- **categories** ((*List[str], optional*). Defaults to *None*.) – The list of object categories in proper order for model training. Example: `['cat','dog','horse']`

**Returns**

`RecordParser` corresponding to the annotation type.

**Return type**

*RecordParser*

**Raises**

**ValueError** – If `annotation_type` is not supported.

**classmethod register**(*annotation\_type: str, parser*) → *None*

Registers a new parser.

**Parameters**

- **annotation\_type** (*str*) – Annotation type which can be `SINGLE_LABEL`, `MULTI_LABEL`, `ENTITY_EXTRACTION` and `BOUNDING_BOX`.
- **parser** (*RecordParser*) – A new Parser class to be registered.

**Returns**

Nothing.

**Return type**

*None*



```
class ads.data_labeling.parser.export_record_parser.SingleLabelRecordParser(dataset_source_path:
                                                                    str, format:
                                                                    Optional[str] =
                                                                    None, categories:
                                                                    Optional[List[str]]
                                                                    = None)
```

Bases: [RecordParser](#)

SingleLabelRecordParser class which parses the label of Single label data.

Initiates a RecordParser instance.

#### Parameters

- **dataset\_source\_path** (*str*) – Dataset source path.
- **format** ((*str*, *optional*). Defaults to None.) – Output format of annotations.
- **categories** ((List[*str*], *optional*). Defaults to None.) – The list of object categories in proper order for model training. Example: ['cat','dog','horse']

#### Returns

RecordParser instance.

#### Return type

[RecordParser](#)

### 18.1.1.5.14 ads.data\_labeling.reader.dataset\_reader module

The module containing classes to read labeled datasets. Allows to read labeled datasets from exports or from the cloud.

#### Classes

##### LabeledDatasetReader

The LabeledDatasetReader class to read labeled dataset.

##### ExportReader

The ExportReader class to read labeled dataset from the export.

##### DLSDatasetReader

The DLSDatasetReader class to read labeled dataset from the cloud.

#### Examples

```
>>> from ads.common import auth as authutil
>>> from ads.data_labeling import LabeledDatasetReader
>>> ds_reader = LabeledDatasetReader.from_export(
...     path="oci://bucket_name@namespace/dataset_metadata.jsonl",
...     auth=authutil.api_keys(),
...     materialize=True
... )
>>> ds_reader.info()
-----
annotation_type                               SINGLE_LABEL
```

(continues on next page)

(continued from previous page)

compartment_id	TEST_COMPARTMENT
dataset_id	TEST_DATASET
dataset_name	test_dataset_name
dataset_type	TEXT
labels	['yes', 'no']
records_path	path/to/records
source_path	path/to/dataset

```
>>> ds_reader.read()
```

	Path	Content	Annotations
0	path/to/the/content/file1	file content	yes
1	path/to/the/content/file2	file content	no
2	path/to/the/content/file3	file content	no

```
>>> next(ds_reader.read(iterator=True))
("path/to/the/content/file1", "file content", "yes")
```

```
>>> next(ds_reader.read(iterator=True, chunksize=2))
[("path/to/the/content/file1", "file content", "yes"),
 ("path/to/the/content/file2", "file content", "no")]
```

```
>>> next(ds_reader.read(chunksize=2))
```

	Path	Content	Annotations
0	path/to/the/content/file1	file content	yes
1	path/to/the/content/file2	file content	no

```
>>> ds_reader = LabeledDatasetReader.from_DLS(
...     dataset_id="dataset_OCID",
...     compartment_id="compartment_OCID",
...     auth=authutil.api_keys(),
...     materialize=True
... )
```

```
class ads.data_labeling.reader.dataset_reader.DLSDatasetReader(dataset_id: str, compartment_id:
                                                                str, auth: Dict, encoding='utf-8',
                                                                materialize: bool = False,
                                                                include_unlabeled: bool = False)
```

Bases: [Reader](#)

The DLSDatasetReader class to read labeled dataset from the cloud.

**info**(self) → [Metadata](#)

Gets the labeled dataset metadata.

**read**(self) → Generator[Tuple, Any, Any]

Reads the labeled dataset.

Initializes the DLS dataset reader instance.

#### Parameters

- **dataset\_id** (str) – The dataset OCID.

- **compartment\_id** (*str*) – The compartment OCID of the dataset.
- **auth** ((*dict*, *optional*). *Defaults to None.*) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **encoding** ((*str*, *optional*). *Defaults to 'utf-8'.*) – Encoding for files. The encoding is used to extract the metadata information of the labeled dataset and also to extract the content of the text dataset records.
- **materialize** ((*bool*, *optional*). *Defaults to False.*) – Whether the content of dataset files should be loaded/materialized or not. By default the content will not be materialized.
- **include\_unlabeled** ((*bool*, *optional*). *Defaults to False.*) – Whether to load the unlabeled records or not.

#### Raises

- **ValueError** – When dataset\_id is empty or not a string.:
- **TypeError** – When dataset\_id not a string.:

**info()** → *Metadata*

Gets the labeled dataset metadata.

#### Returns

The labeled dataset metadata.

#### Return type

*Metadata*

**read**(*format: Optional[str] = None*) → Generator[Tuple, Any, Any]

Reads the labeled dataset records.

#### Parameters

**format** ((*str*, *optional*). *Defaults to None.*) – Output format of annotations. Can be None, “spacy” for dataset Entity Extraction type or “yolo” for Object Detection type. When None, it outputs List[NERItem] or List[BoundingBoxItem]. When “spacy”, it outputs List[Tuple]. When “yolo”, it outputs List[List[Tuple]].

#### Returns

The labeled dataset records.

#### Return type

Generator[Tuple, Any, Any]

```
class ads.data_labeling.reader.dataset_reader.ExportReader(path: str, auth: Optional[Dict] = None,  
                                                         encoding='utf-8', materialize: bool =  
                                                         False, include_unlabeled: bool =  
                                                         False)
```

Bases: *Reader*

The ExportReader class to read labeled dataset from the export.

**info**(*self*) → *Metadata*

Gets the labeled dataset metadata.

**read**(*self*) → Generator[Tuple, Any, Any]

Reads the labeled dataset.

Initializes the labeled dataset export reader instance.

#### Parameters

- **path** (*str*) – The metadata file path, can be either local or object storage path.
- **auth** ((*dict*, *optional*). *Defaults to None.*) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **encoding** ((*str*, *optional*). *Defaults to 'utf-8'.*) – Encoding for files. The encoding is used to extract the metadata information of the labeled dataset and also to extract the content of the text dataset records.
- **materialize** ((*bool*, *optional*). *Defaults to False.*) – Whether the content of dataset files should be loaded/materialized or not. By default the content will not be materialized.
- **include\_unlabeled** ((*bool*, *optional*). *Defaults to False.*) – Whether to load the unlabeled records or not.

#### Raises

- **ValueError** – When path is empty or not a string.:
- **TypeError** – When path not a string.:

**info()** → *Metadata*

Gets the labeled dataset metadata.

#### Returns

The labeled dataset metadata.

#### Return type

*Metadata*

**read**(*format: Optional[str] = None*) → Generator[Tuple, Any, Any]

Reads the labeled dataset records.

#### Parameters

**format** ((*str*, *optional*). *Defaults to None.*) – Output format of annotations. Can be None, “spacy” for dataset Entity Extraction type or “yolo” for Object Detection type. When None, it outputs List[NERItem] or List[BoundingBoxItem]. When “spacy”, it outputs List[Tuple]. When “yolo”, it outputs List[List[Tuple]].

#### Returns

The labeled dataset records.

#### Return type

Generator[Tuple, Any, Any]

**class** ads.data\_labeling.reader.dataset\_reader.**LabeledDatasetReader**(*reader: Reader*)

Bases: object

The labeled dataset reader class.

**info**(*self*) → *Metadata*

Gets labeled dataset metadata.

**read**(*self, iterator: bool = False*) → Union[Generator[Any, Any, Any], pd.DataFrame]

Reads labeled dataset.

```
from_export(cls, path: str, auth: Dict = None, encoding='utf-8', materialize: bool = False) →
    'LabeledDatasetReader'
```

Constructs a Labeled Dataset Reader instance.

## Examples

```
>>> from ads.common import auth as authutil
>>> from ads.data_labeling import LabeledDatasetReader
```

```
>>> ds_reader = LabeledDatasetReader.from_export(
...     path="oci://bucket_name@namespace/dataset_metadata.jsonl",
...     auth=authutil.api_keys(),
...     materialize=True
... )
```

```
>>> ds_reader = LabeledDatasetReader.from_DLS(
...     dataset_id="dataset_OCID",
...     compartment_id="compartment_OCID",
...     auth=authutil.api_keys(),
...     materialize=True
... )
```

```
>>> ds_reader.info()
-----
annotation_type          SINGLE_LABEL
compartment_id           TEST_COMPARTMENT
dataset_id               TEST_DATASET
dataset_name             test_dataset_name
dataset_type             TEXT
labels                   ['yes', 'no']
records_path             path/to/records
source_path              path/to/dataset
```

```
>>> ds_reader.read()
-----
          Path          Content          Annotations
-----
0  path/to/the/content/file1  file content          yes
1  path/to/the/content/file2  file content          no
2  path/to/the/content/file3  file content          no
```

```
>>> next(ds_reader.read(iterator=True))
("path/to/the/content/file1", "file content", "yes")
```

```
>>> next(ds_reader.read(iterator=True, chunksize=2))
[("path/to/the/content/file1", "file content", "yes"),
 ("path/to/the/content/file2", "file content", "no")]
```

```
>>> next(ds_reader.read(chunksize=2))
-----
          Path          Content          Annotations
-----
```

(continues on next page)

(continued from previous page)

0	path/to/the/content/file1	file content	yes
1	path/to/the/content/file2	file content	no

Initializes the labeled dataset reader instance.

#### Parameters

**reader** ([Reader](#)) – The Reader instance which reads and extracts the labeled dataset.

**classmethod from\_DLS**(*dataset\_id: str, compartment\_id: Optional[str] = None, auth: Optional[dict] = None, encoding: str = 'utf-8', materialize: bool = False, include\_unlabeled: bool = False*) → [LabeledDatasetReader](#)

Constructs Labeled Dataset Reader instance.

#### Parameters

- **dataset\_id** (*str*) – The dataset OCID.
- **compartment\_id** (*str*. Defaults to the *compartment\_id* from the env variable.) – The compartment OCID of the dataset.
- **auth** (*(dict, optional)*. Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **encoding** (*(str, optional)*. Defaults to 'utf-8'.) – Encoding for files.
- **materialize** (*(bool, optional)*. Defaults to *False*.) – Whether the content of the dataset file should be loaded or it should return the file path to the content. By default the content will not be loaded.

#### Returns

The LabeledDatasetReader instance.

#### Return type

[LabeledDatasetReader](#)

**classmethod from\_export**(*path: str, auth: Optional[dict] = None, encoding: str = 'utf-8', materialize: bool = False, include\_unlabeled: bool = False*) → [LabeledDatasetReader](#)

Constructs Labeled Dataset Reader instance.

#### Parameters

- **path** (*str*) – The metadata file path, can be either local or object storage path.
- **auth** (*(dict, optional)*. Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **encoding** (*(str, optional)*. Defaults to 'utf-8'.) – Encoding for files.
- **materialize** (*(bool, optional)*. Defaults to *False*.) – Whether the content of the dataset file should be loaded or it should return the file path to the content. By default the content will not be loaded.

#### Returns

The LabeledDatasetReader instance.

#### Return type

[LabeledDatasetReader](#)

**info()** → Serializable

Gets the labeled dataset metadata.

**Returns**

The labeled dataset metadata.

**Return type**

*Metadata*

**read**(*iterator: bool = False, format: Optional[str] = None, chunksize: Optional[int] = None*) →

Union[Generator[Any, Any, Any], DataFrame]

Reads the labeled dataset records.

**Parameters**

- **iterator** (*(bool, optional). Defaults to False.*) – True if the result should be represented as a Generator. False if the result should be represented as a Pandas DataFrame.
- **format** (*(str, optional). Defaults to None.*) – Output format of annotations. Can be None, “spacy” or “yolo”.
- **chunksize** (*(int, optional). Defaults to None.*) – The number of records that should be read in one iteration. The result will be returned in a generator format.

**Returns**

- *Union[ – Generator[Tuple[str, str, Any], Any, Any], Generator[List[Tuple[str, str, Any]], Any, Any], Generator[pd.DataFrame, Any, Any], pd.DataFrame*
- *] – pd.DataFrame if iterator and chunksize are not specified. Generator[pd.DataFrame] `if iterator equal to False and chunksize is specified. Generator[List[Tuple[str, str, Any]]] if iterator equal to True and chunksize is specified. Generator[Tuple[str, str, Any]] if iterator equal to True and chunksize is not specified.*

#### 18.1.1.5.15 ads.data\_labeling.reader.jsonl\_reader module

**class** ads.data\_labeling.reader.jsonl\_reader.JsonlReader(*path: str, auth: Optional[Dict] = None, encoding='utf-8'*)

Bases: *Reader*

JsonlReader class which reads the file.

Initiates a JsonlReader object.

**Parameters**

- **path** (*str*) – object storage path or local path for a file.
- **auth** (*(dict, optional). Defaults to None.*) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **encoding** (*(str, optional). Defaults to 'utf-8'.*) – Encoding of files. Only used for “TEXT” dataset.

## Examples

```
>>> from ads.data_labeling.reader.jsonl_reader import JsonlReader
>>> path = "your/path/to/jsonl/file.jsonl"
>>> from ads.common import auth as authutil
>>> reader = JsonlReader(path=path, auth=authutil.api_keys(), encoding="utf-8")
>>> next(reader.read())
```

**read**(*skip: Optional[int] = None*) → Generator[Dict, Any, Any]

Reads and yields the content of the file.

### Parameters

**skip** ((*int*, *optional*). Defaults to *None*.) – The number of records that should be skipped.

### Returns

The content of the file.

### Return type

Generator[Dict, Any, Any]

### Raises

- **ValueError** – If *skip* not empty and not a positive integer.
- **FileNotFoundError** – When file not found.

## 18.1.1.5.16 ads.data\_labeling.reader.metadata\_reader module

```
class ads.data_labeling.reader.metadata_reader.DLSMetadataReader(dataset_id: str,  
                                                                compartment_id: str, auth:  
                                                                dict)
```

Bases: [Reader](#)

DLSMetadataReader class which reads the metadata jsonl file from the cloud.

Initializes the DLS metadata reader instance.

### Parameters

- **dataset\_id** (*str*) – The dataset OCID.
- **compartment\_id** (*str*) – The compartment OCID of the dataset.
- **auth** (*dict*) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

### Raises

- **ValueError** – When *dataset\_id* is empty or not a string.:
- **TypeError** – When *dataset\_id* not a string.:

**read**() → [Metadata](#)

Reads the content from the metadata file.

### Returns

The metadata of the labeled dataset.



**Return type***Metadata***Raises**

- ***DatasetNotFoundError*** – If dataset not found.
- ***ReadDatasetError*** – If any error occurred in attempt to read dataset.

**exception** `ads.data_labeling.reader.metadata_reader.DatasetNotFoundError(id: str)`

Bases: `Exception`

**exception** `ads.data_labeling.reader.metadata_reader.EmptyMetadata`

Bases: `Exception`

Empty Metadata.

**class** `ads.data_labeling.reader.metadata_reader.ExportMetadataReader(path: str, auth: Optional[Dict] = None, encoding='utf-8')`

Bases: *JsonlReader*

ExportMetadataReader class which reads the metadata jsonl file from local/object storage path.

Initiates a *JsonlReader* object.

**Parameters**

- **path** (*str*) – object storage path or local path for a file.
- **auth** (*(dict, optional). Defaults to None.*) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate `IdentityClient` object.
- **encoding** (*(str, optional). Defaults to 'utf-8'.*) – Encoding of files. Only used for “TEXT” dataset.

**Examples**

```
>>> from ads.data_labeling.reader.jsonl_reader import JsonlReader
>>> path = "your/path/to/jsonl/file.jsonl"
>>> from ads.common import auth as authutil
>>> reader = JsonlReader(path=path, auth=authutil.api_keys(), encoding="utf-8")
>>> next(reader.read())
```

**read()** → *Metadata*

Reads the content from the metadata file.

**Returns**

The metadata of the labeled dataset.

**Return type***Metadata*

**class** `ads.data_labeling.reader.metadata_reader.MetadataReader(reader: Reader)`

Bases: `object`

MetadataReader class which reads and extracts the labeled dataset metadata.

## Examples

```
>>> from ads.data_labeling import MetadataReader
>>> import oci
>>> import os
>>> from ads.common import auth as authutil
>>> reader = MetadataReader.from_export_file("metadata_export_file_path",
...                                         auth=authutil.api_keys())
>>> reader.read()
```

Initiate a MetadataReader instance.

### Parameters

**reader** ([Reader](#)) – Reader instance which reads and extracts the labeled dataset metadata.

**classmethod from\_DLS**(*dataset\_id: str, compartment\_id: Optional[str] = None, auth: Optional[dict] = None*) → [MetadataReader](#)

Constructs a MetadataReader instance.

### Parameters

- **dataset\_id** (*str*) – The dataset OCID.
- **compartment\_id** (*((str, optional). Default None)*) – The compartment OCID of the dataset.
- **auth** (*((dict, optional). Defaults to None.)*) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

### Returns

The MetadataReader instance whose reader is a DLSMetadataReader instance.

### Return type

[MetadataReader](#)

**classmethod from\_export\_file**(*path: str, auth: Optional[Dict] = None*) → [MetadataReader](#)

Constructs a MetadataReader instance.

### Parameters

- **path** (*str*) – metadata file path, can be either local or object storage path.
- **auth** (*((dict, optional). Defaults to None.)*) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

### Returns

The MetadataReader instance whose reader is a ExportMetadataReader instance.

### Return type

[MetadataReader](#)

**read()** → [Metadata](#)

Reads the content from the metadata file.

### Returns

The metadata of the labeled dataset.

**Return type**  
*Metadata*

**exception** `ads.data_labeling.reader.metadata_reader.ReadDatasetError(id: str)`

Bases: Exception

#### 18.1.1.5.17 `ads.data_labeling.reader.record_reader` module

**class** `ads.data_labeling.reader.record_reader.RecordReader(reader: Reader, parser: Parser, loader: Optional[Loader] = None, include_unlabeled: bool = False, encoding: str = 'utf-8', materialize: bool = False)`

Bases: object

Record Reader Class consists of parser, reader and loader. Reader reads the the content from the record file. Parser parses the label for each record. And Loader loads the content of the file path in that record.

#### Examples

```
>>> import os
>>> import oci
>>> from ads.data_labeling import RecordReader
>>> from ads.common import auth as authutil
>>> file_path = "/path/to/your_record.jsonl"
>>> dataset_type = "IMAGE"
>>> annotation_type = "BOUNDING_BOX"
>>> record_reader = RecordReader.from_export_file(file_path, dataset_type,
↪annotation_type, "image_file_path", authutil.api_keys())
>>> next(record_reader.read())
```

Initiates a RecordReader instance.

#### Parameters

- **reader** (`Reader`) – Reader instance to read content from the record file.
- **parser** (`Parser`) – Parser instance to parse the labels from record file.
- **loader** (`Loader`. Defaults to `None`.) – Loader instance to load the content from the file path in the record.
- **materialize** (`bool`, optional. Defaults to `False`.) – Whether to materialize the content using loader.
- **include\_unlabeled** (`bool`, optional). Default to `False`.) – Whether to load the unlabeled records or not.
- **encoding** (`str`, optional) – Encoding for text files. Used only to extract the content of the text dataset contents.

#### Raises

**ValueError** – If the record reader and record parser must be specified. If the loader is not specified when materialize if True.

```
classmethod from_DLS(dataset_id: str, dataset_type: str, annotation_type: str, dataset_source_path: str,
                    compartment_id: Optional[str] = None, auth: Optional[Dict] = None,
                    include_unlabeled: bool = False, encoding: str = 'utf-8', materialize: bool = False,
                    format: Optional[str] = None, categories: Optional[List[str]] = None) →
                    RecordReader
```

Constructs Record Reader instance.

#### Parameters

- **dataset\_id** (*str*) – The dataset OCID.
- **dataset\_type** (*str*) – Dataset type. Currently supports TEXT, IMAGE and DOCUMENT.
- **annotation\_type** (*str*) – Annotation Type. Currently TEXT supports SINGLE\_LABEL, MULTI\_LABEL, ENTITY\_EXTRACTION. IMAGE supports SINGLE\_LABEL, MULTI\_LABEL and BOUNDING\_BOX. DOCUMENT supports SINGLE\_LABEL and MULTI\_LABEL.
- **dataset\_source\_path** (*str*) – Dataset source path.
- **compartment\_id** ((*str*, *optional*). Defaults to None.) – The compartment OCID of the dataset.
- **auth**((*dict*, *optional*). Defaults to None.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **encoding**((*str*, *optional*). Defaults to 'utf-8'.) – Encoding for files.
- **materialize**((*bool*, *optional*). Defaults to False.) – Whether the content of the dataset file should be loaded or it should return the file path to the content. By default the content will not be loaded.
- **format** ((*str*, *optional*). Defaults to None.) – Output format of annotations. Can be None, “spacy” for dataset Entity Extraction type or “yolo” for Object Detection type. When None, it outputs List[NERItem] or List[BoundingBoxItem]. When “spacy”, it outputs List[Tuple]. When “yolo”, it outputs List[List[Tuple]].
- **categories** ((*List[str]*, *optional*). Defaults to None.) – The list of object categories in proper order for model training. Example: ['cat','dog','horse']

#### Returns

The RecordReader instance.

#### Return type

*RecordReader*

```
classmethod from_export_file(path: str, dataset_type: str, annotation_type: str, dataset_source_path:
                             str, auth: Optional[Dict] = None, include_unlabeled: bool = False,
                             encoding: str = 'utf-8', materialize: bool = False, format: Optional[str]
                             = None, categories: Optional[List[str]] = None,
                             includes_metadata=False) → RecordReader
```

Initiates a RecordReader instance.

#### Parameters

- **path** (*str*) – Record file path.
- **dataset\_type** (*str*) – Dataset type. Currently supports TEXT, IMAGE and DOCUMENT.

- **annotation\_type** (*str*) – Annotation Type. Currently TEXT supports SINGLE\_LABEL, MULTI\_LABEL, ENTITY\_EXTRACTION. IMAGE supports SINGLE\_LABEL, MULTI\_LABEL and BOUNDING\_BOX. DOCUMENT supports SINGLE\_LABEL and MULTI\_LABEL.
- **dataset\_source\_path** (*str*) – Dataset source path.
- **auth** ((*dict*, *optional*). *Default None*) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **include\_unlabeled** ((*bool*, *optional*). *Default to False.*) – Whether to load the unlabeled records or not.
- **encoding** ((*str*, *optional*). *Defaults to "utf-8".*) – Encoding for text files. Used only to extract the content of the text dataset contents.
- **materialize** ((*bool*, *optional*). *Defaults to False.*) – Whether to materialize the content by loader.
- **format** ((*str*, *optional*). *Defaults to None.*) – Output format of annotations. Can be None, “spacy” for dataset Entity Extraction type or “yolo” for Object Detection type. When None, it outputs List[NERItem] or List[BoundingBoxItem]. When “spacy”, it outputs List[Tuple]. When “yolo”, it outputs List[List[Tuple]].
- **categories** ((*List[str]*, *optional*). *Defaults to None.*) – The list of object categories in proper order for model training. Example: ['cat','dog','horse']
- **includes\_metadata** ((*bool*, *optional*). *Defaults to False.*) – Determines whether the export file includes metadata or not.

**Returns**

A RecordReader instance.

**Return type**

*RecordReader*

**read()** → Generator[Tuple[str, Union[List, str]], Any, Any]

Reads the record.

**Yields**

*Generator[Tuple[str, Union[List, str]], Any, Any]* – File path, content and labels in a tuple.

**18.1.1.5.18 ads.data\_labeling.visualizer.image\_visualizer module**

The module that helps to visualize Image Dataset.

**ads.data\_labeling.visualizer.image\_visualizer.render**(*items: List[LabeledImageItem]*, *options: Dict = None*)

Renders Labeled Image dataset.

## Examples

```
>>> bbox1 = BoundingBoxItem(bottom_left=(0.3, 0.4),
>>>                           top_left=(0.3, 0.09),
>>>                           top_right=(0.86, 0.09),
>>>                           bottom_right=(0.86, 0.4),
>>>                           labels=['dolphin', 'fish'])
```

```
>>> record1 = LabeledImageItem(img_obj1, [bbox1])
```

```
>>> bbox2 = BoundingBoxItem(bottom_left=(0.2, 0.4),
>>>                           top_left=(0.2, 0.2),
>>>                           top_right=(0.8, 0.2),
>>>                           bottom_right=(0.8, 0.4),
>>>                           labels=['dolphin'])
>>> bbox3 = BoundingBoxItem(bottom_left=(0.5, 1.0),
>>>                           top_left=(0.5, 0.8),
>>>                           top_right=(0.8, 0.8),
>>>                           bottom_right=(0.8, 1.0),
>>>                           labels=['shark'])
```

```
>>> record2 = LabeledImageItem(img_obj2, [bbox2, bbox3])
>>> render(items = [record1, record2], options={"default_color":"blue", "colors": {
↪ "dolphin":"blue", "whale":"red"}})
```

**class** `ads.data_labeling.visualizer.image_visualizer.ImageLabeledDataFormatter`

Bases: `object`

The ImageRender class to render Image items in a notebook session.

**static render\_item**(*item: LabeledImageItem, options: Optional[Dict] = None, path: Optional[str] = None*) → *None*

Renders image dataset.

### Parameters

- **item** (`LabeledImageItem`) – Item to render.
- **options** (`Optional[dict]`) – Render options.
- **path** (`str`) – Path to save the image with annotations to local directory.

### Returns

Nothing.

### Return type

`None`

### Raises

- **ValueError** – If items not provided. If path is not valid.
- **TypeError** – If items provided in a wrong format.

**class** `ads.data_labeling.visualizer.image_visualizer.LabeledImageItem`(*img: ImageFile, boxes: List[BoundingBoxItem]*)

Bases: `object`

Data class representing Image Item.

**img**

the labeled image object.

**Type**

ImageFile

**boxes**

a list of BoundingBoxItem

**Type**List[*BoundingBoxItem*]**boxes:** List[*BoundingBoxItem*]**img:** ImageFile

```
class ads.data_labeling.visualizer.image_visualizer.RenderOptions(default_color: str, colors: Optional[dict])
```

Bases: object

Data class representing render options.

**default\_color**

The specified default color.

**Type**

str

**colors**

The multiple specified colors.

**Type**

Optional[dict]

**colors:** Optional[dict]**default\_color:** str**classmethod from\_dict**(*options: dict*) → *RenderOptions*

Constructs an instance of RenderOptions from a dictionary.

**Parameters****options** (*dict*) – Render options in dictionary format.**Returns**

The instance of RenderOptions.

**Return type***RenderOptions***to\_dict()**

Converts RenderOptions instance to dictionary format.

**Returns**

The render options in dictionary format.

**Return type**

dict

**exception** ads.data\_labeling.visualizer.image\_visualizer.WrongEntityFormat

Bases: ValueError

`ads.data_labeling.visualizer.image_visualizer.render`(*items*: List[LabeledImageItem], *options*: Optional[Dict] = None, *path*: Optional[str] = None) → None

Render image dataset.

#### Parameters

- **items** (List[LabeledImageItem]) – The list of LabeledImageItem to render.
- **options** (dict, optional) – The options for rendering.
- **path** (str) – Path to save the images with annotations to local directory.

#### Returns

Nothing.

#### Return type

None

#### Raises

- **ValueError** – If items not provided. If path is not valid.
- **TypeError** – If items provided in a wrong format.

### Examples

```
>>> bbox1 = BoundingBoxItem(bottom_left=(0.3, 0.4),
>>>                             top_left=(0.3, 0.09),
>>>                             top_right=(0.86, 0.09),
>>>                             bottom_right=(0.86, 0.4),
>>>                             labels=['dolphin', 'fish'])
```

```
>>> record1 = LabeledImageItem(img_obj1, [bbox1])
>>> render(items = [record1])
```

#### 18.1.1.5.19 ads.data\_labeling.visualizer.text\_visualizer module

The module that helps to visualize NER Text Dataset.

`ads.data_labeling.visualizer.text_visualizer.render`(*items*: List[LabeledTextItem], *options*: Dict = None) → str

Renders NER dataset to Html format.

### Examples

```
>>> record1 = LabeledTextItem("London is the capital of the United Kingdom", [NERItem(
↪ 'city', 0, 6), NERItem("country", 29, 14)])
>>> record2 = LabeledTextItem("Houston area contractor seeking a Sheet Metal_
↪ Superintendent.", [NERItem("city", 0, 6)])
>>> result = render(items = [record1, record2], options={"default_color": "#DDEECC",
↪ "colors": {"city": "#DDEECC", "country": "#FFAAAA"}})
>>> display(HTML(result))
```



```
class ads.data_labeling.visualizer.text_visualizer.LabeledTextItem(txt: str, ents:
                                                                    List[NERItem])

    Bases: object
    Data class representing NER Item.

    txt
        The labeled sentence.

        Type
            str

    ents
        The list of entities.

        Type
            List[NERItem]

    ents: List[NERItem]

    txt: str
```

```
class ads.data_labeling.visualizer.text_visualizer.RenderOptions(default_color: str, colors:
                                                                    Optional[dict])

    Bases: object
    Data class representing render options.

    default_color
        The specified default color.

        Type
            str

    colors
        The multiple specified colors.

        Type
            Optional[dict]

    colors: Optional[dict]

    default_color: str

    classmethod from_dict(options: dict) → RenderOptions
        Constructs an instance of RenderOptions from a dictionary.

        Parameters
            options (dict) – Render options in dictionary format.

        Returns
            The instance of RenderOptions.

        Return type
            RenderOptions

    to_dict()
        Converts RenderOptions instance to dictionary format.

        Returns
            The render options in dictionary format.
```

**Return type**

dict

**class** ads.data\_labeling.visualizer.text\_visualizer.**TextLabeledDataFormatter**

Bases: object

The TextLabeledDataFormatter class to render NER items into Html format.

**static render**(items: List[LabeledTextItem], options: Optional[Dict] = None) → str

Renders NER dataset to Html format.

**Parameters**

- **items** (List[LabeledTextItem]) – Items to render.
- **options** (Optional[dict]) – Render options.

**Returns**

Html representation of rendered NER dataset.

**Return type**

str

**Raises**

- **ValueError** – If items not provided.
- **TypeError** – If items provided in a wrong format.

**ads.data\_labeling.visualizer.text\_visualizer.render**(items: List[LabeledTextItem], options: Optional[Dict] = None) → str

Renders NER dataset to Html format.

**Parameters**

- **items** (List[LabeledTextItem]) – The list of NER items to render.
- **options** (dict, optional) – The options for rendering.

**Returns**

Html string.

**Return type**

str

**Examples**

```
>>> record = LabeledTextItem("London is the capital of the United Kingdom",  
↳ [NERItem('city', 0, 6), NERItem("country", 29, 14)])  
>>> result = render(items = [record], options={"default_color": "#DDEECC", "colors":  
↳ {"city": "#DDEECC", "country": "#FFAAAA"}})  
>>> display(HTML(result))
```

#### 18.1.1.5.20 Module contents

### 18.1.1.6 ads.database package

#### 18.1.1.6.1 Subpackages

#### 18.1.1.6.2 Submodules

#### 18.1.1.6.3 ads.database.connection module

**class** ads.database.connection.**Connector**(*secret\_id: Optional[str] = None, key: Optional[str] = None, repository\_path: Optional[str] = None, \*\*kwargs*)

Bases: object

Validate that a connection could be made for the given set of connection parameters, and construct a Connector object provided that the validation is successful.

##### Parameters

- **secret\_id** (*str, optional*) – The ocid of the secret to retrieve from Oracle Cloud Infrastructure Vault.
- **key** (*str, optional*) – The key to find the database directory.
- **repository\_path** (*str, optional*) – The local database information store, default to `~/.database` unless specified otherwise.
- **kwargs** (*dict, optional*) – Name-value pairs that are to be added to the list of connection parameters. For example, `database_name="mydb"`, `database_type="oracle"`, `username = "root"`, `password = "pwd"`.

##### Return type

A Connector object.

**connect()**

**class** ads.database.connection.**OracleConnector**(*oracle\_connection\_config*)

Bases: object

ads.database.connection.**get\_repository**(*key: str, repository\_path: Optional[str] = None*) → dict

Get all values from local database store.

##### Parameters

- **key** (*str*) – The key to find the database directory.
- **repository\_path** (*str, optional*) – The path to local database store, default to `~/.database` unless specified otherwise.

##### Return type

A dictionary of all values in the store.

ads.database.connection.**import\_wallet**(*wallet\_path: str, key: str, repository\_path: Optional[str] = None*) → None

Saves wallet to local database store. Unzip the wallet zip file, update sqlnet.ora and store wallet files.

##### Parameters

- **wallet\_path** (*str*) – The local path to the downloaded wallet zip file.

- **key** (*str*) – The key to find the database directory.
- **repository\_path** (*str*, *optional*) – The local database store, default to ~/.database unless specified otherwise.

`ads.database.connection.update_repository(value: dict, key: str, replace: bool = True, repository_path: Optional[str] = None) → dict`

Saves value into local database store.

#### Parameters

- **value** (*dict*) – The values to store locally.
- **key** (*str*) – The key to find the local database directory.
- **replace** (*bool*, *default to True*) – If set to false, updates the stored value.
- **repository\_path** (*str: str*, *optional*) – The local database store, default to ~/.database unless specified otherwise.

#### Return type

A dictionary of all values in the repository for the given key.

### 18.1.1.6.4 Module contents

### 18.1.1.7 ads.dataflow package

#### 18.1.1.7.1 Submodules

#### 18.1.1.7.2 ads.dataflow.dataflow module

```
class ads.dataflow.dataflow.DataFlow(compartment_id=None,
                                     dataflow_base_folder='/home/datascience/dataflow', os_auth=None,
                                     df_auth=None)
```

Bases: object

**create\_app**(*app\_config: dict*, *overwrite\_script=False*, *overwrite\_archive=False*) → object

Create a new dataflow application with the supplied app config. *app\_config* contains parameters needed to create a new application, according to `oci.data_flow.models.CreateApplicationDetails`.

#### Parameters

- **app\_config** (*dict*) – the config file that contains all necessary parameters used to create a dataflow app
- **overwrite\_script** (*bool*) – whether to overwrite the existing pyscript script on Object Storage
- **overwrite\_archive** (*bool*) – whether to overwrite the existing archive file on Object Storage

#### Returns

**df\_app** – New dataflow application.

#### Return type

`oci.dataflow.models.Application`

**get\_app**(*app\_id: str*)

Get the Project based on app\_id.

**Parameters**

**app\_id** (*str*, *required*) – The OCID of the dataflow app to get.

**Returns**

**app** – The oci.dataflow.models.Application with the matching ID.

**Return type**

oci.dataflow.models.Application

**list\_apps**(*include\_deleted: bool = False*, *compartment\_id: Optional[str] = None*, *datetime\_format: str = '%Y-%m-%d %H:%M:%S'*, *\*\*kwargs*) → object

List all apps in a given compartment, or in the current notebook session's compartment.

**Parameters**

- **include\_deleted** (*bool*, *optional*, *default=False*) – Whether to include deleted apps in the returned list.
- **compartment\_id** (*str*, *optional*, *default: NB\_SESSION\_COMPARTMENT\_OCID*) – The compartment specified to list apps.
- **datetime\_format** (*str*, *optional*, *default: '%Y-%m-%d %H:%M:%S'*) – Change format for date time fields.

**Returns**

**dsl** – List of Dataflow applications.

**Return type**

List

**load\_app**(*app\_id: str*, *target\_folder: Optional[str] = None*) → object

Load an existing dataflow application based on application id. The existing dataflow application can be created either from dataflow service or the dataflow integration of ADS.

**Parameters**

- **app\_id** (*str*, *required*) – The OCID of the dataflow app to load.
- **target\_folder** (*str*, *optional*,) – the folder to store the local artifacts of this application. If not specified, the target\_folder will use the dataflow\_base\_folder by default.

**Returns**

**dfa** – A dataflow application of type ads.dataflow.dataflow.DataFlowApp

**Return type**

ads.dataflow.dataflow.DataFlowApp

**prepare\_app**(*display\_name: str*, *script\_bucket: str*, *pyspark\_file\_path: str*, *spark\_version: str = '2.4.4'*, *compartment\_id: Optional[str] = None*, *archive\_path: Optional[str] = None*, *archive\_bucket: Optional[str] = None*, *logs\_bucket: str = 'dataflow-logs'*, *driver\_shape: str = 'VM.Standard2.4'*, *executor\_shape: str = 'VM.Standard2.4'*, *num\_executors: int = 1*, *arguments: list = []*, *script\_parameters: dict = {}*) → dict

Check if the parameters provided by users to create an application are valid and then prepare app\_configuration for creating an app or saving for future reuse.

**Parameters**

- **display\_name** (*str*, *required*) – A user-friendly name. This name is not necessarily unique.

- **script\_bucket** (*str*, *required*) – bucket in object storage to upload the pyspark file
- **pyspark\_file\_path** (*str*, *required*) – path to the pyspark file
- **spark\_version** (*str*) – Allowed values are “2.4.4”, “3.0.2”.
- **compartment\_id** (*str*) – OCID of the compartment to create a dataflow app. If not provided, compartment\_id will use the same as the notebook session.
- **archive\_path** (*str*, *optional*) – path to the archive file
- **archive\_bucket** (*str*, *optional*) – bucket in object storage to upload the archive file
- **logs\_bucket** (*str*, *default is 'dataflow-logs'*) – bucket in object storage to put run logs
- **driver\_shape** (*str*) – The value to assign to the driver\_shape property of this CreateApplicationDetails. Allowed values for this property are: “VM.Standard2.1”, “VM.Standard2.2”, “VM.Standard2.4”, “VM.Standard2.8”, “VM.Standard2.16”, “VM.Standard2.24”.
- **executor\_shape** (*str*) – The value to assign to the executor\_shape property of this CreateApplicationDetails. Allowed values for this property are: “VM.Standard2.1”, “VM.Standard2.2”, “VM.Standard2.4”, “VM.Standard2.8”, “VM.Standard2.16”, “VM.Standard2.24”.
- **num\_executors** (*int*) – The number of executor VMs requested.
- **arguments** (*list of str*) – The values passed into the command line string to run the application
- **script\_parameters** (*dict*) – The value of the parameters passed to the running application as command line arguments for the pyspark script.

**Returns****app\_configuration****Return type**

dictionary containing all the validated params for CreateApplicationDetails.

**template**(*job\_type: str = 'standard\_pyspark', script\_str: str = '', file\_dir: Optional[str] = None, file\_name: Optional[str] = None*) → *str*

Populate a prewritten pyspark or sparksql python script with user’s choice to write additional lines and save in local directory.

**Parameters**

- **job\_type** (*str*, *default is 'standard\_pyspark'*) – Currently supports two types, ‘standard\_pyspark’ or ‘sparksql’
- **script\_str** (*str*, *optional*, *default is ''*) – code provided by user to write in the python script
- **file\_dir** (*str*, *optional*) – Directory to save the python script in local directory
- **file\_name** (*str*, *optional*) – name of the python script to save to the local directory

**Returns****script\_path** – Path to the template generated python file in local directory**Return type***str*

**class** `ads.dataflow.dataflow.DataFlowApp`(*app\_config*, *app\_response*, *app\_dir*, *oci\_link*, *\*\*kwargs*)

Bases: `DataFlow`

**property config: dict**

Retrieve the app\_config file used to create the data flow app

**Returns**

**app\_config** – dictionary containing all the validated params for this DataFlowApp

**Return type**

Dict

**get\_run**(*run\_id: str*)

Get the Run based on run\_id

**Parameters**

**run\_id** (*str*, *required*) – The OCID of the dataflow run to get.

**Returns**

**df\_run** – The oci.dataflow.models.Run with the matching ID.

**Return type**

oci.dataflow.models.Run

**list\_runs**(*include\_failed: bool = False*, *datetime\_format: str = '%Y-%m-%d %H:%M:%S'*, *\*\*kwargs*) → object

List all run of a dataflow app

**Parameters**

- **include\_failed** (*bool*, *optional*, *default=False*) – Whether to include failed runs in the returned list
- **datetime\_format** (*str*, *optional*, *default: '%Y-%m-%d %H:%M:%S'*) – Change format for date time fields

**Returns**

**df\_runs** – List of Data flow runs.

**Return type**

List

**property oci\_link: object**

Retrieve the oci link of the data flow app

**Returns**

**oci\_link** – a link to the app page in an oci console.

**Return type**

str

**prepare\_run**(*run\_display\_name: str*, *compartment\_id: Optional[str] = None*, *logs\_bucket: str = ''*, *driver\_shape: str = 'VM.Standard2.4'*, *executor\_shape: str = 'VM.Standard2.4'*, *num\_executors: int = 1*, *\*\*kwargs*) → dict

Check if the parameters provided by users to create a run are valid and then prepare run\_config for creating run details.

**Parameters**

- **run\_display\_name** (*str*) – A user-friendly name. This name is not necessarily unique.
- **compartment\_id** (*str*) – OCID of the compartment to create a dataflow run. If not provided, compartment\_id will use the same as the dataflow app.

- **logs\_bucket** (*str*) – bucket in object storage to put run logs, if not provided, will use the same logs\_bucket as defined in app\_config
- **driver\_shape** (*str*) – The value to assign to the driver\_shape property of this CreateApplicationDetails. Allowed values for this property are: “VM.Standard2.1”, “VM.Standard2.2”, “VM.Standard2.4”, “VM.Standard2.8”, “VM.Standard2.16”, “VM.Standard2.24”.
- **executor\_shape** (*str*) – The value to assign to the executor\_shape property of this CreateApplicationDetails. Allowed values for this property are: “VM.Standard2.1”, “VM.Standard2.2”, “VM.Standard2.4”, “VM.Standard2.8”, “VM.Standard2.16”, “VM.Standard2.24”.
- **num\_executors** (*int*) – The number of executor VMs requested.

**Returns**

**run\_config** – Dictionary containing all the validated params for CreateRunDetails.

**Return type**

Dict

**run**(*run\_config: dict, save\_log\_to\_local: bool = False, copy\_script\_to\_object\_storage: bool = True, copy\_archive\_to\_object\_storage: bool = True, pyspark\_file\_path: Optional[str] = None, archive\_path: Optional[str] = None, wait: bool = True*) → object

Create a new dataflow run with the supplied run config. run\_config contains parameters needed to create a new run, according to oci.data\_flow.models.CreateRunDetails.

**Parameters**

- **run\_config** (*dict, required*) – The config file that contains all necessary parameters used to create a dataflow run
- **save\_log\_to\_local** (*bool, optional*) – A boolean value that defaults to false. If set to true, it saves the log files to local dir
- **copy\_script\_to\_object\_storage** (*bool, optional*) – A boolean value that defaults to true. Local script will be copied to object storage
- **copy\_archive\_to\_object\_storage** (*bool, optional*) – A boolean value that defaults to true. Local archive file will be copied to object storage
- **pyspark\_file\_path** (*str, optional*) – The pyspark file path used for creating the dataflow app. if pyspark\_file\_path isn’t specified then reuse the path that the app was created with.
- **archive\_path** (*str, optional*) – The archive file path used for creating the dataflow app. if archive\_path isn’t specified then reuse the path that the app was created with.
- **wait** (*bool, optional*) – A boolean value that defaults to true. When True, the return will be ads.dataflow.dataflow.DataFlowRun in terminal state. When False, the return will be a ads.dataflow.dataflow.RunObserver.

**Returns**

**df\_run** – Either a new Data Flow run or a run observer.

**Return type**

Variable

**class** ads.dataflow.dataflow.DataFlowLog(*text, oci\_path, log\_local\_dir*)

Bases: object



**head**(*n: int = 10*)

Show the first *n* lines of the log as the output of the notebook cell

**Parameters**

**n**(*int, default is 10*) – the number of lines from head of the log file

**Return type**

None

**property local\_dir**

Get the local directory where the log file is saved.

**Returns**

**local\_dir** – Path to the local directory where the log file is saved.

**Return type**

str

**property local\_path**

Get the path of the log file in local directory

**Returns**

**local\_path** – Path of the log file in local directory

**Return type**

str

**property oci\_path**

Get the path of the log file in object storage

**Returns**

**oci\_path** – Path of the log file in object storage

**Return type**

str

**save**(*log\_dir=None*)

save the log file to a local directory.

**Parameters**

- **log\_dir** (*str*,) – The path to the local directory to save log file, if not
- **set** –
- **default.** (*log will be saved to the \_local\_dir by*) –

**Return type**

None

**show\_all**()

Show all content of the log as the output of the notebook cell

**Return type**

None

**tail**(*n: int = 10*)

Show the last *n* lines of the log as the output of the notebook cell

**Parameters**

**n**(*int, default is 10*) – the number of lines from tail of the log file

**Return type**

None

```
class ads.dataflow.dataflow.DataFlowRun(run_config, run_response, save_log_to_local, local_dir,
                                         **kwargs)
```

Bases: *DataFlow***LOG\_OUTPUTS** = ['stdout', 'stderr']**property config: dict**

Retrieve the run\_config file used to create the Data Flow run

**Returns****run\_config** – dictionary containing all the validated params for this DataFlowRun**Return type**

Dict

**fetch\_log(log\_type: str) → object**

Fetch the log information of a run

**Parameters****log\_type** (*str*, have two values, 'stdout' or 'stderr') –**Returns****dfl** – a Data Flow log object**Return type***DataFlowLog***property local\_dir: str**

Retrieve the local directory of the data flow run

**Returns****local\_dir** – the local path to the Data Flow run**Return type**

str

**property log\_stderr: object**

Retrieve the stderr of the data flow run

**Returns****log\_error** – a clickable link that opens the stderr log in another tab in jupyter notebook environment**Return type***ads.dataflow.dataflow.DataFlowLog***property log\_stdout: object**

Retrieve the stdout of the data flow run

**Returns****log\_out** – a clickable link that opens the stdout log in another tab in a JupyterLab notebook environment**Return type***ads.dataflow.dataflow.DataFlowLog*

**property oci\_link: object**

Retrieve the oci link of the data flow run

**Returns**

**oci\_link** – link to the run page in an oci console

**Return type**

str

**property status: str**

Retrieve the status of the data flow run

**Returns**

**status** – String that describes the status of the run

**Return type**

str

**update\_config(param\_dict) → None**

Modify the run\_config file used to create the data flow run

**Parameters**

**param\_dict** (*Dict*) – Dictionary containing the key value pairs of the run\_config parameters and the updated values.

**Return type**

None

**class ads.dataflow.dataflow.RunObserver(app, run\_config, save\_log\_to\_local)**

Bases: object

**property config: dict**

Retrieve the run\_config file used to create the data flow run

**Returns**

**run\_config** – Dictionary containing all the validated parameters for this Data Flow run

**Return type**

Dict

**property local\_dir: str**

Retrieve the local directory of the data flow run

**Returns**

**local\_dir** – the local path to the Data Flow run

**Return type**

str

**property oci\_link: object**

Retrieve the oci link of the data flow run

**Returns**

**oci\_link** – link to the run page in an oci console

**Return type**

str

**property status: str**

Returns the lifecycle state of the Data Flow run

**update\_config**(*param\_dict*) → None

Modify the run\_config file used to create the data flow run

**Parameters**

**param\_dict** (*Dict*) – dictionary containing the key value pairs of the run\_config parameters and the updated values.

**Return type**

None

**wait()**

Wait and monitor the run creation process.

**Parameters**

**None** –

**Returns**

**df\_run** – The oci.dataflow.models.Run after monitoring is done.

**Return type**

oci.dataflow.models.Run

**class** ads.dataflow.dataflow.**SPARK\_VERSION**

Bases: str

**v2\_4\_4** = '2.4.4'

**v3\_0\_2** = '3.0.2'

### 18.1.1.7.3 ads.dataflow.dataflowssummary module

**class** ads.dataflow.dataflowssummary.**SummaryList**(*entity\_list, datetime\_format='%Y-%m-%d %H:%M:%S'*)

Bases: list

**abstract filter**(*selection, instance=None*)

Abstract filter method for dataflow summary.

**abstract sort\_by**(*columns, reverse=False*)

Abstract sort method for dataflow summary.

**to\_dataframe**(*datetime\_format=None*)

Abstract to\_dataframe method for dataflow summary.

### 18.1.1.7.4 Module contents

### 18.1.1.8 ads.dataset package

#### 18.1.1.8.1 Submodules

#### 18.1.1.8.2 ads.dataset.classification\_dataset module

```
class ads.dataset.classification_dataset.BinaryClassificationDataset(df, sampled_df, target,
                                                                    target_type, shape,
                                                                    positive_class=None,
                                                                    **kwargs)
```

Bases: *ClassificationDataset*

Dataset for binary classification

```
set_positive_class(positive_class, missing_value=False)
```

Return new dataset with values in target column mapped to True or False in accordance with the specified positive label.

#### Parameters

- **positive\_class** (same dtype as target) – The target label which should be identified as positive outcome from model.
- **missing\_value** (bool) – missing values will be converted to this

#### Returns

dataset

#### Return type

same type as the caller

#### Raises

*ValidationError* – if the positive\_class is not present in target

### Examples

```
>>> ds = DatasetFactory.open("iris.csv")
>>> ds_with_target = ds.set_target('class')
>>> ds_with_pos_class = ds.set_positive_class('setosa')
```

```
class ads.dataset.classification_dataset.BinaryTextClassificationDataset(df, sampled_df,
                                                                    target, target_type,
                                                                    shape, **kwargs)
```

Bases: *BinaryClassificationDataset*

Dataset for binary text classification

```
auto_transform()
```

Automatically chooses the most effective dataset transformation

```
select_best_features(score_func=None, k=12)
```

Automatically chooses the best features and removes the rest

```
class ads.dataset.classification_dataset.ClassificationDataset(df, sampled_df, target, target_type,
                                                                    shape, **kwargs)
```

Bases: *ADSDatasetWithTarget*

Dataset for classification task

```
auto_transform(fix_imbalance: bool = True, correlation_threshold: float = 0.7, frac: float = 1.0,
                correlation_methods: str = 'pearson')
```

Return transformed dataset with several optimizations applied automatically. The optimizations include:

- Dropping constant and primary key columns, which has no predictive quality,

- Imputation, to fill in missing values in noisy data:
  - For continuous variables, fill with mean if less than 40% is missing, else drop,
  - For categorical variables, fill with most frequent if less than 40% is missing, else drop,
- Dropping strongly co-correlated columns that tend to produce less generalizable models,
- Balancing dataset using up or down sampling.

#### Parameters

- **fix\_imbalance** (*bool, defaults to True.*) – Fix imbalance between classes in dataset. Used only for classification datasets.
- **correlation\_threshold** (*float, defaults to 0.7. It must be between 0 and 1, inclusive.*) – The correlation threshold where columns with correlation higher than the threshold will be considered as strongly co-correlated and recommended to be taken care of.
- **frac** (*float, defaults to 1.0. Range -> (0, 1].*) – What fraction of the data should be used in the calculation?
- **correlation\_methods** (*Union[list, str], defaults to 'pearson'.*) –
  - ‘pearson’: Use Pearson’s Correlation between continuous features,
  - ‘cramers v’: Use Cramer’s V correlations between categorical features,
  - ‘correlation ratio’: Use Correlation Ratio Correlation between categorical and continuous features,
  - ‘all’: Is equivalent to [‘pearson’, ‘cramers v’, ‘correlation ratio’].Or a list containing any combination of these methods, for example, [‘pearson’, ‘cramers v’].

#### Returns

**transformed\_dataset** – The dataset after transformation

#### Return type

*ADSDatasetWithTarget*

### Examples

```
>>> ds_clean = ds.auto_transform(correlation_threshold=0.6)
```

**convert\_to\_text\_classification**(*text\_column: str*)

Builds a new dataset with the given text column as the only feature besides target.

#### Parameters

**text\_column** (*str*) – Feature name to use for text classification task

#### Returns

**ds** – Dataset with one text feature and a classification target

#### Return type

TextClassificationDataset

## Examples

```
>>> review_ds = DatasetFactory.open("review_data.csv")
>>> ds_text_class = review_ds.convert_to_text_classification('reviews')
```

### `down_sample(sampler=None)`

Fixes an imbalanced dataset by down-sampling.

#### Parameters

**sampler** (An instance of *SamplerMixin*) – Should implement `fit_resample(X,y)` method. If `None`, does random down sampling.

#### Returns

**down\_sampled\_ds** – A down-sampled dataset.

#### Return type

*ClassificationDataset*

## Examples

```
>>> ds = DatasetFactory.open("some_data.csv")
>>> ds_balanced_small = ds.down_sample()
```

### `up_sample(sampler='default')`

Fixes imbalanced dataset by up-sampling

#### Parameters

- **sampler** (An instance of *SamplerMixin*) – Should implement `fit_resample(X,y)` method. If 'default', either SMOTE or random sampler will be used
- **fill\_missing\_type** (a string) – Can either be 'mean', 'mode' or 'median'.

#### Returns

**up\_sampled\_ds** – an up-sampled dataset

#### Return type

*ClassificationDataset*

## Examples

```
>>> ds = DatasetFactory.open("some_data.csv")
>>> ds_balanced_large = ds.up_sample()
```

```
class ads.dataset.classification_dataset.MultiClassClassificationDataset(df, sampled_df,
                                                                    target, target_type,
                                                                    shape, **kwargs)
```

Bases: *ClassificationDataset*

Dataset for multi-class classification

```
class ads.dataset.classification_dataset.MultiClassTextClassificationDataset(df, sampled_df,
                                                                    target,
                                                                    target_type,
                                                                    shape,
                                                                    **kwargs)
```

Bases: *MultiClassClassificationDataset*

Dataset for multi-class text classification

**auto\_transform()**

Automatically chooses the most effective dataset transformation

**select\_best\_features**(*score\_func=None, k=12*)

Automatically chooses the best features and removes the rest

### 18.1.1.8.3 ads.dataset.correlation module

### 18.1.1.8.4 ads.dataset.correlation\_plot module

**class** ads.dataset.correlation\_plot.**BokehHeatMap**(*ds*)

Bases: object

Generate a HeatMap or horizontal bar plot to compare features.

**debug()**

Return True if in debug mode, otherwise False.

**flatten\_corr\_matrix**(*corr\_matrix*)

Flatten a correlation matrix into a pandas Dataframe.

**Parameters**

**corr\_matrix** (*Pandas Dataframe*) – The correlation matrix to be flattened.

**Returns**

**corr\_flatten** – The flattened correlation matrix.

**Return type**

Pandas DataFrame

**generate\_heatmap**(*corr\_matrix, title: str, msg: str, correlation\_threshold: float*)

Generate a heatmap from a correlation matrix.

**Parameters**

- **corr\_matrix** (*Pandas Dataframe*) – The dataframe to be used for heatmap generation.
- **title** (*str*) – title of the heatmap.
- **msg** (*str*) – An additional msg to include in the plot.
- **correlation\_threshold** (*float*) – A float between 0 and 1 which is used for excluding correlations which are not intense enough from the plot.

**Returns**

**tab** – A matplotlib Panel object which includes a plotted heatmap

**Return type**

matplotlib Panel

**generate\_target\_heatmap**(*corr\_matrix, title: str, correlation\_target: str, msg: str, correlation\_threshold: float*)

Generate a heatmap from a correlation matrix and its targets.

**Parameters**



- **corr\_matrix** (*Pandas Dataframe*) – The dataframe to be used for heatmap generation.
- **title** (*str*) – title of the heatmap.
- **correlation\_target** (*str*) – The target column name for computing correlations against.
- **msg** (*str*) – An additional msg to include in the plot.
- **correlation\_threshold** (*float*) – A float between 0 and 1 which is used for excluding correlations which are not intense enough from the plot.

**Returns**

**tab** – A matplotlib Panel object which includes a plotted heatmap.

**Return type**

matplotlib Panel

**plot\_correlation\_heatmap**(*ds*, *plot\_type: str = 'heatmap'*, *correlation\_target: str = None*, *correlation\_threshold=-1*, *correlation\_methods: str = 'pearson'*, *\*\*kwargs*)

Plots a correlation heatmap.

**Parameters**

- **ds** (*Pandas Slice*) – A data slice or file
- **plot\_type** (*str Defaults to "heatmap"*) – The type of plot - “bar” is another option.
- **correlation\_target** (*str, Defaults to None*) – the target column for correlation calculations.
- **correlation\_threshold** (*float, Defaults to -1*) – the threshold for computing correlation heatmap elements.
- **correlation\_methods** (*str, Defaults to "pearson"*) – the way to compute correlations, other options are “cramers v” and “correlation ratio”

**plot\_hbar**(*matrix*, *low: float = 1*, *high=1*, *title: str = None*, *tool\_tips: list = None*, *column\_name: str = None*)

Plots a histogram bar-graph.

**Parameters**

- **matrix** (*Pandas Dataframe*) – The dataframe to be plotted.
- **low** (*float, Defaults to 1*) – The color mapping value for “low” points.
- **high** (*float, Defaults to 1*) – The color mapping value for “high” points.
- **title** (*str, Defaults to None*) – The optional title of the heat map.
- **tool\_tips** (*list of str, Defaults to None*) – An optional list of tool tips to include with the plot.
- **column\_name** (*str, Defaults to None*) – The name of the column which is being plotted.

**Returns**

**fig** – A matplotlib heatmap figure object.

**Return type**

matplotlib Figure

**plot\_heat\_map**(*matrix*, *xrange*: list, *yrange*: list, *low*: float = 1, *high*=1, *title*: str = None, *tool\_tips*: list = None)

Plots a matrix as a heatmap.

**Parameters**

- **matrix** (*Pandas Dataframe*) – The dataframe to be plotted.
- **xrange** (*List of floats*) – The range of x values to plot.
- **yrange** (*List of floats*) – The range of y values to plot.
- **low** (*float, Defaults to 1*) – The color mapping value for “low” points.
- **high** (*float, Defaults to 1*) – The color mapping value for “high” points.
- **title** (*str, Defaults to None*) – The optional title of the heat map.
- **tool\_tips** (*list of str, Defaults to None*) – An optional list of tool tips to include with the plot.

**Returns**

**fig** – A matplotlib heatmap figure object.

**Return type**

matplotlib Figure

`ads.dataset.correlation_plot.plot_correlation_heatmap(ds=None, **kwargs) → None`

Plots a correlation heatmap.

**Parameters**

**ds** (*Pandas Slice*) – A data slice or file

#### 18.1.1.8.5 ads.dataset.dask\_series module

#### 18.1.1.8.6 ads.dataset.dataframe\_transformer module

**class** `ads.dataset.dataframe_transformer.DataFrameTransformer`(*func\_name*, *target\_name*,  
*target\_sample\_val*, *args*=None,  
*kw\_args*=None)

Bases: `TransformerMixin`

A DataFrameTransformer object.

**fit**(*df*)

Takes in a DF and returns a fitted model

**transform**(*df*)

Takes in a DF and returns a transformed DF

`ads.dataset.dataframe_transformer.expand_lambda_function(lambda_func)`

Returns a lambda function after expansion.

### 18.1.1.8.7 ads.dataset.dataset module

```
class ads.dataset.dataset.ADSDataset(df, sampled_df, shape, name="", description=None,
                                     type_discovery=True, types={}, metadata=None,
                                     progress=<ads.dataset.progress.DummyProgressBar object>,
                                     transformer_pipeline=None, interactive=False, **kwargs)
```

Bases: *PandasDataset*

An ADSDataset Object.

The ADSDataset object cannot be used for classification or regression problems until a target has been set using *set\_target*. To see some rows in the data use any of the usual Pandas functions like *head()*. There are also a variety of converters, to\_dask, to\_pandas, to\_h2o, to\_xgb, to\_csv, to\_parquet, to\_json & to\_hdf .

**assign\_column**(column, arg)

Return new dataset with new column or values of the existing column mapped according to input correspondence.

Used for adding a new column or substituting each value in a column with another value, that may be derived from a function, a *pandas.Series* or a *pandas.DataFrame*.

#### Parameters

- **column** (*str*) – Name of the feature to update.
- **arg** (*function, dict, Series or DataFrame*) – Mapping correspondence.

#### Returns

**dataset** – a dataset with the specified column assigned.

#### Return type

same type as the caller

### Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_same_size = ds.assign_column('target', lambda x: x>15 if x not None)
>>> ds_bigger = ds.assign_column('new_col', np.arange(ds.shape[0]))
```

**astype**(types)

Convert data type of features.

#### Parameters

**types** (*dict*) – key is the existing feature name value is the data type to which the values of the feature should be converted. Valid data types: All numpy datatypes (Example: *np.float64*, *np.int64*, ...) or one of categorical, continuous, ordinal or datetime.

#### Returns

**updated\_dataset** – an ADSDataset with new data types

#### Return type

*ADSDataset*

## Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_reformatted = ds.astype({"target": "categorical"})
```

**call**(*func*, \**args*, *sample\_size*=None, \*\**kwargs*)

Runs a custom function on dataframe

*func* will receive the pandas dataframe (which represents the dataset) as an argument named ‘df’ by default. This can be overridden by specifying the dataframe argument name in a tuple (*func*, *dataframe\_name*).

### Parameters

- **func** (*Union[callable, tuple]*) – Custom function that takes pandas dataframe as input Alternatively a (callable, data) tuple where data is a string indicating the keyword of callable that expects the dataframe name
- **args** (*iterable, optional*) – Positional arguments passed into func
- **sample\_size** (*int, Optional*) – To use a sampled dataframe
- **kwargs** (*mapping, optional*) – A dictionary of keyword arguments passed into func

### Returns

**func** – a plotting function that contains \**args* and \*\**kwargs*

### Return type

function

## Examples

```
>>> ds = DatasetFactory.open("classification_data.csv")
>>> def f1(df):
...     return(sum(df), axis=0)
>>> sum_ds = ds.call(f1)
```

**compute**()

**corr**(*correlation\_methods*: *Union[list, str]* = ‘pearson’, *frac*: *float* = 1.0, *sample\_size*: *float* = 1.0, *nan\_threshold*: *float* = 0.8, *overwrite*: *Optional[bool]* = None, *force\_recompute*: *bool* = False)

Compute pairwise correlation of numeric and categorical columns, output a matrix or a list of matrices computed using the correlation methods passed in.

### Parameters

- **correlation\_methods** (*Union[list, str]*, default to ‘pearson’) –
  - ‘pearson’: Use Pearson’s Correlation between continuous features,
  - ‘cramers v’: Use Cramer’s V correlations between categorical features,
  - ‘correlation ratio’: Use Correlation Ratio Correlation between categorical and continuous features,
  - ‘all’: Is equivalent to [‘pearson’, ‘cramers v’, ‘correlation ratio’].

Or a list containing any combination of these methods, for example, [‘pearson’, ‘cramers v’].

- **frac** – Is deprecated and replaced by *sample\_size*.

- **sample\_size** (*float, defaults to 1.0. Float, Range -> (0, 1]*) – What fraction of the data should be used in the calculation?
- **nan\_threshold** (*float, default to 0.8, Range -> [0, 1]*) – Only compute a correlation when the proportion of the values, in a column, is less than or equal to `nan_threshold`.
- **overwrite** – Is deprecated and replaced by `force_recompute`.
- **force\_recompute** (*bool, default to be False*) –
  - If `False`, it calculates the correlation matrix if there is no cached correlation matrix. Otherwise, it returns the cached correlation matrix.
  - If `True`, it calculates the correlation matrix regardless whether there is cached result or not.

**Returns**

**correlation** – The pairwise correlations as a matrix (`DataFrame`) or list of matrices

**Return type**

`Union[list, pandas.DataFrame]`

**property ddf**

`df_read_functions = ['head', 'describe', '_get_numeric_data']`

**drop\_columns(columns)**

Return new dataset with specified columns removed.

**Parameters**

**columns** (*str or list*) – columns to drop.

**Returns**

**dataset** – a dataset with specified columns dropped.

**Return type**

same type as the caller

**Raises**

**`ValidationError`** – If any of the feature names is not found in the dataset.

**Examples**

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_smaller = ds.drop_columns(['col1', 'col2'])
```

**merge(data, \*\*kwargs)**

Merges this dataset with another `ADSDataset` or `pandas dataframe`.

**Parameters**

- **data** (*`Union[ADSDataset, pandas.DataFrame]`*) – Data to merge.
- **kwargs** (*`dict, optional`*) – additional keyword arguments that would be passed to underlying dataframe's merge API.

### Examples

```
>>> ds1 = DatasetFactory.open("data1.csv")
>>> ds2 = DatasetFactory.open("data2.csv")
>>> ds_12 = ds1.merge(ds2)
```

#### **rename\_columns**(*columns*)

Returns a new dataset with altered column names.

dict values must be unique (1-to-1). Labels not contained in a dict will be left as-is. Extra labels listed don't throw an error.

##### **Parameters**

**columns** (*dict-like or function or list of str*) – dict to rename columns selectively, or list of names to rename all columns, or a function like str.upper

##### **Returns**

**dataset** – A dataset with specified columns renamed.

##### **Return type**

same type as the caller

### Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_renamed = ds.rename_columns({'col1': 'target'})
```

#### **sample**(*frac=None, random\_state=42*)

Returns random sample of dataset.

##### **Parameters**

- **frac** (*float, optional*) – Fraction of axis items to return.
- **random\_state** (int or np.random.RandomState) – If int we create a new RandomState with this as the seed Otherwise we draw from the passed RandomState

##### **Returns**

**sampled\_dataset** – An ADSDataset which was randomly sampled.

##### **Return type**

*ADSDataset*

### Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_sample = ds.sample()
```

#### **set\_description**(*description*)

Sets description for the dataset.

Give your dataset a description.

##### **Parameters**

**description** (*str*) – Description of the dataset.

## Examples

```
>>> ds = DatasetFactory.open("data1.csv")
>>> ds_renamed = ds.set_description("dataset1 is from \"data1.csv\"")
```

### set\_name(*name*)

Sets name for the dataset.

This name will be used to filter the datasets returned by `ds.list()` API. Calling this API is optional. By default name of the dataset is set to empty.

#### Parameters

**name** (*str*) – Name of the dataset.

## Examples

```
>>> ds = DatasetFactory.open("data1.csv")
>>> ds_renamed = ds.set_name("dataset1")
```

### set\_target(*target*, *type\_discovery=True*, *target\_type=None*)

Returns a dataset tagged based on the type of target.

#### Parameters

- **target** (*str*) – name of the feature to use as target.
- **type\_discovery** (*bool*) – This is set as True by default.
- **target\_type** (*type*) – If provided, then the target will be typed with the provided value.

#### Returns

**ds** – tagged according to the type of the target column.

#### Return type

*ADSDataset*

## Examples

```
>>> ds = DatasetFactory.open("classification_data.csv")
>>> ds_with_target= ds.set_target("target_class")
```

**show\_corr**(*frac: float = 1.0*, *sample\_size: float = 1.0*, *nan\_threshold: float = 0.8*, *overwrite: Optional[bool] = None*, *force\_recompute: bool = False*, *correlation\_target: Optional[str] = None*, *plot\_type: str = 'heatmap'*, *correlation\_threshold: float = -1*, *correlation\_methods='pearson'*, *\*\*kwargs*)

Show heatmap or barplot of pairwise correlation of numeric and categorical columns, output three tabs which are heatmap or barplot of correlation matrix of numeric columns vs numeric columns using pearson correlation method, categorical columns vs categorical columns using Cramer's V method, and numeric vs categorical columns, excluding NA/null values and columns which have more than 80% of NA/null values. By default, only 'pearson' correlation is calculated and shown in the first tab. Set `correlation_methods='all'` to show all correlation charts.

#### Parameters

- **frac** (*Is superseded by sample\_size*) –

- **sample\_size** (*float, defaults to 1.0. Float, Range -> (0, 1]*) – What fraction of the data should be used in the calculation?
- **nan\_threshold** (*float, defaults to 0.8, Range -> [0, 1]*) – In the default case, it will only calculate the correlation of the columns which has less than or equal to 80% of missing values.
- **overwrite** – Is deprecated and replaced by `force_recompute`.
- **force\_recompute** (*bool, default to be False.*) –
  - If False, it calculates the correlation matrix if there is no cached correlation matrix. Otherwise, it returns the cached correlation matrix.
  - If True, it calculates the correlation matrix regardless whether there is cached result or not.
- **plot\_type** (*str, default to "heatmap"*) – It can only be “heatmap” or “bar”. Note that if “bar” is chosen, `correlation_target` also has to be set and the bar chart will only show the correlation values of the pairs which have the target in them.
- **correlation\_target** (*str, default to None*) – It can be any columns of type continuous, ordinal, categorical or zipcode. When `correlation_target` is set, only pairs that contains `correlation_target` will show.
- **correlation\_threshold** (*float, default to -1*) – It can be any number between -1 and 1.
- **correlation\_methods** (*Union[list, str], defaults to 'pearson'*) –
  - ‘pearson’: Use Pearson’s Correlation between continuous features,
  - ‘cramers v’: Use Cramer’s V correlations between categorical features,
  - ‘correlation ratio’: Use Correlation Ratio Correlation between categorical and continuous features,
  - ‘all’: Is equivalent to [‘pearson’, ‘cramers v’, ‘correlation ratio’].Or a list containing any combination of these methods, for example, [‘pearson’, ‘cramers v’].

**Return type**

None

**show\_in\_notebook**(*correlation\_threshold=-1, selected\_index=0, sample\_size=0, visualize\_features=True, correlation\_methods='pearson', \*\*kwargs*)

Provide visualization of dataset.

- Display feature distribution. The data table display will show a maximum of 8 digits,
- Plot the correlation between the dataset features (as a heatmap) only when all the features are continuous or ordinal,
- Display data head.

**Parameters**

- **correlation\_threshold** (*int, default -1*) – The correlation threshold to select, which only show features that have larger or equal correlation values than the threshold.
- **selected\_index** (*int, str, default 0*) – The displayed output is stacked into an accordion widget, use `selected_index` to force the display to open a specific element, use the (zero offset) index or any prefix string of the name (eg, ‘corr’ for correlations)



- **sample\_size** (*int*, *default 0*) – The size (in rows) to sample for visualizations
- **visualize\_features** (*bool default False*) – For the “Features” section control if feature visualizations are shown or not. If not only a summary of the numeric statistics is shown. The numeric statistics are also always shows for wide (>64 features) datasets
- **correlation\_methods** (*Union[list, str]*, *default to 'pearson'*) –
  - ‘pearson’: Use Pearson’s Correlation between continuous features,
  - ‘cramers v’: Use Cramer’s V correlations between categorical features,
  - ‘correlation ratio’: Use Correlation Ratio Correlation between categorical and continuous features,
  - ‘all’: Is equivalent to [‘pearson’, ‘cramers v’, ‘correlation ratio’].
 Or a list containing any combination of these methods, for example, [‘pearson’, ‘cramers v’].

**snapshot** (*snapshot\_dir=None, name="", storage\_options=None*)

Snapshot the dataset with modifications made so far.

Optionally caller can invoke `ds.set_name()` before saving to identify the dataset uniquely at the time of using `ds.list()`.

The snapshot can be reloaded by providing the URI returned by this API to `DatasetFactory.open()`

#### Parameters

- **snapshot\_dir** (*str, optional*) – Directory path under which dataset snapshot will be created. Defaults to `snapshots_dir` set using `DatasetFactory.set_default_storage()`.
- **name** (*str, optional, default: ""*) – Name to uniquely identify the snapshot using `DatasetFactory.list_snapshots()`. If not provided, an auto-generated name is used.
- **storage\_options** (*dict, optional*) – Parameters passed on to the backend filesystem class. Defaults to `storage_options` set using `DatasetFactory.set_default_storage()`.

#### Returns

**p\_str** – the URI to access the snapshotted dataset.

#### Return type

str

### Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_uri = ds.snapshot()
```

**to\_avro** (*path, schema=None, storage\_options=None, \*\*kwargs*)

Save data to Avro files. Avro is a remote procedure call and data serialization framework developed within Apache’s Hadoop project. It uses JSON for defining data types and protocols, and serializes data in a compact binary format.

#### Parameters

- **path** (*string*) – Path to a target filename. May contain a \* to denote many filenames.
- **schema** (*dict*) – Avro schema dictionary, see below.

- **storage\_options** (*dict*, *optional*) – Parameters passed to the backend filesystem class. Defaults to storage\_options set using DatasetFactory.set\_default\_storage().
- **kwargs** (*dict*, *optional*) – See <https://fastavro.readthedocs.io/en/latest/writer.html>

## Notes

Avro schema is a complex dictionary describing the data, see <https://avro.apache.org/docs/1.8.2/gettingstartedpython.html#Defining+a+schema> and <https://fastavro.readthedocs.io/en/latest/writer.html>. Its structure is as follows:

```
{'name': 'Test',
 'namespace': 'Test',
 'doc': 'Descriptive text',
 'type': 'record',
 'fields': [
   {'name': 'a', 'type': 'int'},
 ]}
```

where the “name” field is required, but “namespace” and “doc” are optional descriptors; “type” must always be “record”. The list of fields should have an entry for every key of the input records, and the types are like the primitive, complex or logical types of the Avro spec (<https://avro.apache.org/docs/1.8.2/spec.html>).

## Examples

```
>>> ds = DatasetFactory.open("data.avro")
>>> ds.to_avro("my/path.avro")
```

**to\_csv**(*path*, *storage\_options=None*, *\*\*kwargs*)

Save the materialized dataframe to csv file.

### Parameters

- **path** (*str*) – Location to write to. If there are more than one partitions in df, should include a glob character to expand into a set of file names, or provide a *name\_function=parameter*. Supports protocol specifications such as “oci://”, “s3://”.
- **storage\_options** (*dict*, *optional*) – Parameters passed on to the backend filesystem class. Defaults to storage\_options set using DatasetFactory.set\_default\_storage().
- **kwargs** (*dict*, *optional*) –

## Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> [ds_link] = ds.to_csv("my/path.csv")
```

**to\_dask**(*filter=None*, *frac=None*, *npartitions=None*, *include\_transformer\_pipeline=False*)

Returns a copy of the data as dask.dataframe.core.DataFrame, and a sklearn pipeline optionally that holds the transformations run so far on the data.

The pipeline returned can be updated with the transformations done offline and passed along with the dataframe to Dataset.open API if the transformations need to be reproduced at the time of scoring.

### Parameters

- **filter** (*str, optional*) – The query string to filter the dataframe, for example `ds.to_dask(filter="age > 50 and location == 'san francisco'")` See also <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.query.html>
- **frac** (*float, optional*) – fraction of original data to return.
- **include\_transformer\_pipeline** (*bool, default: False*) – If True, (dataframe, transformer\_pipeline) is returned as a tuple.

#### Returns

- **dataframe** (*dask.dataframe.core.DataFrame*) – if `include_transformer_pipeline` is False.
- **(data, transformer\_pipeline)** (*tuple of dask.dataframe.core.DataFrame and dataset.pipeline.TransformerPipeline*) – if `include_transformer_pipeline` is True.

#### Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_dask = ds.to_dask()
```

#### Notes

See also <http://docs.dask.org/en/latest/dataframe-api.html#dataframe> and <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline>

**to\_dask\_dataframe**(*filter=None, frac=None, npartitions=None, include\_transformer\_pipeline=False*)

**to\_h2o**(*filter=None, frac=None, include\_transformer\_pipeline=False*)

Returns a copy of the data as `h2o.H2OFrame`, and a sklearn pipeline optionally that holds the transformations run so far on the data.

The pipeline returned can be updated with the transformations done offline and passed along with the dataframe to Dataset.open API if the transformations need to be reproduced at the time of scoring.

#### Parameters

- **filter** (*str, optional*) – The query string to filter the dataframe, for example `ds.to_h2o(filter="age > 50 and location == 'san francisco'")` See also <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.query.html>
- **frac** (*float, optional*) – fraction of original data to return.
- **include\_transformer\_pipeline** (*bool, default: False*) – If True, (dataframe, transformer\_pipeline) is returned as a tuple.

#### Returns

- **dataframe** (*h2o.H2OFrame*) – if `include_transformer_pipeline` is False.
- **(data, transformer\_pipeline)** (*tuple of h2o.H2OFrame and dataset.pipeline.TransformerPipeline*) – if `include_transformer_pipeline` is True.

## Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_as_h2o = ds.to_h2o()
```

## Notes

See also <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline>

**to\_h2o\_dataframe**(*filter=None, frac=None, include\_transformer\_pipeline=False*)

**to\_hdf**(*path: str, key: str, storage\_options: Optional[dict] = None, \*\*kwargs*) → str

Save data to Hierarchical Data Format (HDF) files.

### Parameters

- **path** (*string*) – Path to a target filename.
- **key** (*string*) – Datapath within the files.
- **storage\_options** (*dict, optional*) – Parameters passed to the backend filesystem class. Defaults to `storage_options` set using `DatasetFactory.set_default_storage()`.
- **kwargs** (*dict, optional*) –

### Returns

The filename of the HDF5 file created.

### Return type

str

## Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds.to_hdf(path="my/path.h5", key="df")
```

**to\_json**(*path, storage\_options=None, \*\*kwargs*)

Save data to JSON files.

### Parameters

- **path** (*str*) – Location to write to. If there are more than one partitions in `df`, should include a glob character to expand into a set of file names, or provide a `name_function=parameter`. Supports protocol specifications such as “`oci://`”, “`s3://`”.
- **storage\_options** (*dict, optional*) – Parameters passed on to the backend filesystem class. Defaults to `storage_options` set using `DatasetFactory.set_default_storage()`.
- **kwargs** (*dict, optional*) –

## Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds.to_json("my/path.json")
```

**to\_pandas**(*filter=None, frac=None, include\_transformer\_pipeline=False*)

Returns a copy of the data as `pandas.DataFrame`, and a sklearn pipeline optionally that holds the transformations run so far on the data.

The pipeline returned can be updated with the transformations done offline and passed along with the dataframe to `Dataset.open` API if the transformations need to be reproduced at the time of scoring.

### Parameters

- **filter** (*str, optional*) – The query string to filter the dataframe, for example `ds.to_pandas(filter="age > 50 and location == 'san francisco'")` See also <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.query.html>
- **frac** (*float, optional*) – fraction of original data to return.
- **include\_transformer\_pipeline** (*bool, default: False*) – If True, (dataframe, transformer\_pipeline) is returned as a tuple

### Returns

- **dataframe** (*pandas.DataFrame*) – if `include_transformer_pipeline` is False.
- (**data**, **transformer\_pipeline**) (*tuple of pandas.DataFrame and dataset.pipeline.TransformerPipeline*) – if `include_transformer_pipeline` is True.

## Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds_as_df = ds.to_pandas()
```

## Notes

See also <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline>

**to\_pandas\_dataframe**(*filter=None, frac=None, include\_transformer\_pipeline=False*)

**to\_parquet**(*path, storage\_options=None, \*\*kwargs*)

Save data to parquet file.

### Parameters

- **path** (*str*) – Location to write to. If there are more than one partitions in df, should include a glob character to expand into a set of file names, or provide a `name_function=parameter`. Supports protocol specifications such as `"oci://"`, `"s3://"`.
- **storage\_options** (*dict, optional*) – Parameters passed on to the backend filesystem class. Defaults to storage\_options set using `DatasetFactory.set_default_storage()`.
- **kwargs** (*dict, optional*) –

## Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> ds.to_parquet("my/path")
```

**to\_xgb**(*filter=None, frac=None, include\_transformer\_pipeline=False*)

Returns a copy of the data as `xgboost.DMatrix`, and a sklearn pipeline optionally that holds the transformations run so far on the data.

The pipeline returned can be updated with the transformations done offline and passed along with the dataframe to `Dataset.open` API if the transformations need to be reproduced at the time of scoring.

### Parameters

- **filter** (*str, optional*) – The query string to filter the dataframe, for example `ds.to_xgb(filter="age > 50 and location == 'san francisco'")` See also <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.query.html>
- **frac** (*float, optional*) – fraction of original data to return.
- **include\_transformer\_pipeline** (*bool, default: False*) – If True, (dataframe, transformer\_pipeline) is returned as a tuple.

### Returns

- **dataframe** (*xgboost.DMatrix*) – if `include_transformer_pipeline` is False.
- (**data**, **transformer\_pipeline**) (*tuple of xgboost.DMatrix and dataset.pipeline.TransformerPipeline*) – if `include_transformer_pipeline` is True.

## Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> xgb_dmat = ds.to_xgb()
```

## Notes

See also <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline>

**to\_xgb\_dmatrix**(*filter=None, frac=None, include\_transformer\_pipeline=False*)

### 18.1.1.8.8 ads.dataset.dataset\_browser module

**class** `ads.dataset.dataset_browser.DatasetBrowser`

Bases: `ABC`

**static** `GitHub(user: str, repo: str, branch: str = 'master')`

Returns a `GitHubDataset`

**static** `filesystem(folder: str)`

Returns a `LocalFilesystemDataset`.

**filter\_list**(*L, filter\_pattern*)  $\rightarrow$  `List[str]`

Filters a list of dataset names.

```
static list(filter_pattern='*') → List[str]
```

Return a list of dataset browser strings.

```
abstract open(**kwargs)
```

Return new dataset for the given name.

**Parameters**

**name** (*str*) – the name of the dataset to open.

**Returns**

**ds**

**Return type**

Dataset

### Examples

```
ds_browser = DatasetBrowser("sklearn")
```

```
ds = ds_browser.open("iris")
```

```
static seaborn()
```

Returns a SeabornDataset.

```
static sklearn()
```

Returns a SklearnDataset.

```
static web(index_url: str)
```

Returns a WebDataset.

```
class ads.dataset.dataset_browser.GitHubDatasets(user: str, repo: str, branch: str)
```

Bases: [DatasetBrowser](#)

```
list(filter_pattern: str = '.*') → List[str]
```

Return a list of dataset browser strings.

```
open(name: str, **kwargs)
```

Return new dataset for the given name.

**Parameters**

**name** (*str*) – the name of the dataset to open.

**Returns**

**ds**

**Return type**

Dataset

### Examples

```
ds_browser = DatasetBrowser("sklearn")
```

```
ds = ds_browser.open("iris")
```

```
class ads.dataset.dataset_browser.LocalFilesystemDatasets(folder: str)
```

Bases: [DatasetBrowser](#)

**list**(*filter\_pattern: str = '.\*')* → List[str]  
Return a list of dataset browser strings.

**open**(*name: str, \*\*kwargs*)  
Return new dataset for the given name.

**Parameters**

**name** (*str*) – the name of the dataset to open.

**Returns**

**ds**

**Return type**

Dataset

### Examples

```
ds_browser = DatasetBrowser("sklearn")
ds = ds_browser.open("iris")
```

**class** ads.dataset.dataset\_browser.**SeabornDatasets**

Bases: [\*DatasetBrowser\*](#)

**list**(*filter\_pattern: str = '.\*')* → List[str]  
Return a list of dataset browser strings.

**open**(*name: str, \*\*kwargs*)  
Return new dataset for the given name.

**Parameters**

**name** (*str*) – the name of the dataset to open.

**Returns**

**ds**

**Return type**

Dataset

### Examples

```
ds_browser = DatasetBrowser("sklearn")
ds = ds_browser.open("iris")
```

**class** ads.dataset.dataset\_browser.**SklearnDatasets**

Bases: [\*DatasetBrowser\*](#)

**list**(*filter\_pattern: str = '.\*')* → List[str]  
Return a list of dataset browser strings.

**open**(*name: str, \*\*kwargs*)  
Return new dataset for the given name.

**Parameters**

**name** (*str*) – the name of the dataset to open.

**Returns**

**ds**



**Return type**

Dataset

**Examples**

```
ds_browser = DatasetBrowser("sklearn")
```

```
ds = ds_browser.open("iris")
```

```
sklearn_datasets = ['breast_cancer', 'diabetes', 'iris', 'wine', 'digits']
```

```
class ads.dataset.dataset_browser.WebDatasets(index_url: str)
```

Bases: [DatasetBrowser](#)

```
list(filter_pattern: str = '.*') → List[str]
```

Return a list of dataset browser strings.

```
open(name: str, **kwargs)
```

Return new dataset for the given name.

**Parameters**

**name** (*str*) – the name of the dataset to open.

**Returns**

**ds**

**Return type**

Dataset

**Examples**

```
ds_browser = DatasetBrowser("sklearn")
```

```
ds = ds_browser.open("iris")
```

**18.1.1.8.9 ads.dataset.dataset\_with\_target module**

```
class ads.dataset.dataset_with_target.ADSDataSetWithTarget(df, sampled_df, target, target_type,
                                                            shape, sample_max_rows=-1,
                                                            type_discovery=True, types={},
                                                            parent=None, name="",
                                                            metadata=None,
                                                            transformer_pipeline=None,
                                                            description=None,
                                                            progress=<ads.dataset.progress.DummyProgressBar
                                                            object>, **kwargs)
```

Bases: [ADSDataSet](#)

This class provides APIs for preparing dataset for modeling.

```
auto_transform(correlation_threshold: float = 0.7, frac: float = 1.0, sample_size=1.0,
               correlation_methods: Union[str, list] = 'pearson')
```

Return transformed dataset with several optimizations applied automatically. The optimizations include:

- Dropping constant and primary key columns, which has no predictive quality,
- Imputation, to fill in missing values in noisy data:

- For continuous variables, fill with mean if less than 40% is missing, else drop,
- For categorical variables, fill with most frequent if less than 40% is missing, else drop,
- Dropping strongly co-correlated columns that tend to produce less generalizable models.

#### Parameters

- **correlation\_threshold** (*float, defaults to 0.7. It must be between 0 and 1, inclusive*) – the correlation threshold where columns with correlation higher than the threshold will be considered as strongly co-correlated and recommended to be taken care of.
- **frac** (*Is superseded by sample\_size*) –
- **sample\_size** (*float, defaults to 1.0. Float, Range -> (0, 1]*) – What fraction of the data should be used in the calculation?
- **correlation\_methods** (*Union[list, str], defaults to 'pearson'*) –
  - 'pearson': Use Pearson's Correlation between continuous features,
  - 'cramers v': Use Cramer's V correlations between categorical features,
  - 'correlation ratio': Use Correlation Ratio Correlation between categorical and continuous features,
  - 'all': Is equivalent to ['pearson', 'cramers v', 'correlation ratio'].Or a list containing any combination of these methods, for example, ['pearson', 'cramers v'].

#### Returns

**transformed\_dataset**

#### Return type

*ADSDatasetWithTarget*

#### Examples

```
>>> ds_clean = ds.auto_transform()
```

```
get_recommendations(correlation_methods: str = 'pearson', correlation_threshold: float = 0.7, frac: float = 1.0, sample_size: float = 1.0, overwrite: bool = None, force_recompute: bool = False, display_format: str = 'widget')
```

Generate recommendations for dataset optimization. This includes:

- Identifying constant and primary key columns, which has no predictive quality,
- Imputation, to fill in missing values in noisy data:
  - For continuous variables, fill with mean if less than 40% is missing, else drop,
  - For categorical variables, fill with most frequent if less than 40% is missing, else drop,
- Identifying strongly co-correlated columns that tend to produce less generalizable models,
- Automatically balancing dataset for classification problems using up or down sampling.

#### Parameters

- **correlation\_methods** (*Union[list, str], default to 'pearson'*) –

- ‘pearson’: Use Pearson’s Correlation between continuous features,
- ‘cramers v’: Use Cramer’s V correlations between categorical features,
- ‘correlation ratio’: Use Correlation Ratio Correlation between categorical and continuous features,
- ‘all’: Is equivalent to [‘pearson’, ‘cramers v’, ‘correlation ratio’].

Or a list containing any combination of these methods, for example, [‘pearson’, ‘cramers v’].

- **correlation\_threshold** (*float, defaults to 0.7. It must be between 0 and 1, inclusive*) – The correlation threshold where columns with correlation higher than the threshold will be considered as strongly co-correlated and recommended to be taken care of.
- **frac** (*Is superseded by sample\_size*) –
- **sample\_size** (*float, defaults to 1.0. Float, Range -> (0, 1]*) – What fraction of the data should be used in the calculation?
- **overwrite** – Is deprecated and replaced by `force_recompute`.
- **force\_recompute** (*bool, default to be False*) –
  - If False, it calculates the correlation matrix if there is no cached correlation matrix. Otherwise, it returns the cached correlation matrix.
  - If True, it calculates the correlation matrix regardless whether there is cached result or not.
- **display\_format** (*string, defaults to 'widget'.*) – Should be either ‘widget’ or ‘table’. If ‘widget’, a GUI style interface is popped out; if ‘table’, a table of suggestions is shown.

#### **get\_transformed\_dataset()**

Return the transformed dataset with the recommendations applied.

This method should be called after applying the recommendations using the `Recommendation#show_in_notebook()` API.

#### **rename\_columns(columns)**

Returns a dataset with columns renamed.

#### **select\_best\_features(score\_func=None, k=12)**

Return new dataset containing only the top k features.

##### **Parameters**

- **k** (*int, default 12*) – The top ‘k’ features to select.
- **score\_func** (*function*) – Scoring function to use to rank the features. This scoring function should take a 2d array X(features) and an array like y(target) and return a numeric score for each feature in the same order as X.

## Notes

See also [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_selection.f\\_regression.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html) and [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_selection.f\\_classif.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html)

## Examples

```
>>> ds = DatasetBrowser("sklearn").open("iris")
>>> ds_small = ds.select_best_features(k=2)
```

**suggest\_recommendations**(*correlation\_methods*: Union[str, list] = 'pearson', *print\_code*: bool = True, *correlation\_threshold*: float = 0.7, *overwrite*: Optional[bool] = None, *force\_recompute*: bool = False, *frac*: float = 1.0, *sample\_size*: float = 1.0, *\*\*kwargs*)

Returns a pandas dataframe with suggestions for dataset optimization. This includes:

- Identifying constant and primary key columns, which has no predictive quality,
- Imputation, to fill in missing values in noisy data:
  - For continuous variables, fill with mean if less than 40% is missing, else drop,
  - For categorical variables, fill with most frequent if less than 40% is missing, else drop,
- Identifying strongly co-correlated columns that tend to produce less generalizable models,
- Automatically balancing dataset for classification problems using up or down sampling.

### Parameters

- **correlation\_methods** (Union[list, str], default to 'pearson') –
  - 'pearson': Use Pearson's Correlation between continuous features,
  - 'cramers v': Use Cramer's V correlations between categorical features,
  - 'correlation ratio': Use Correlation Ratio Correlation between categorical and continuous features,
  - 'all': Is equivalent to ['pearson', 'cramers v', 'correlation ratio'].

Or a list containing any combination of these methods, for example, ['pearson', 'cramers v']

- **print\_code** (bool, Defaults to True) – Print Python code for the suggested actions.
- **correlation\_threshold** (float. Defaults to 0.7. It must be between 0 and 1, inclusive) – the correlation threshold where columns with correlation higher than the threshold will be considered as strongly co-correlated and recommended to be taken care of.
- **frac** (Is superseded by sample\_size) –
- **sample\_size** (float, defaults to 1.0. Float, Range -> (0, 1]) – What fraction of the data should be used in the calculation?
- **overwrite** – Is deprecated and replaced by force\_recompute.
- **force\_recompute** (bool, default to be False) –
  - If False, it calculates the correlation matrix if there is no cached correlation matrix. Otherwise, it returns the cached correlation matrix.

- If True, it calculates the correlation matrix regardless whether there is cached result or not.

**Returns****suggestion dataframe****Return type**

pandas.DataFrame

**Examples**

```
>>> suggestion_df = ds.suggest_recommendations(correlation_threshold=0.7)
```

**train\_test\_split**(*test\_size=0.1, random\_state=42*)

Splits dataset to train and test data.

**Parameters**

- **test\_size** (*Union[float, int], optional, default=0.1*) –
- **random\_state** (*Union[int, RandomState], optional, default=None*) –
  - If int, random\_state is the seed used by the random number generator;
  - If RandomState instance, random\_state is the random number generator;
  - If None, the random number generator is the RandomState instance used by np.random.

**Returns****train\_data, test\_data** – tuple of ADSData instances**Return type**

tuple

**Examples**

```
>>> ds = DatasetFactory.open("data.csv")
>>> train, test = ds.train_test_split()
```

**train\_validation\_test\_split**(*test\_size=0.1, validation\_size=0.1, random\_state=42*)

Splits dataset to train, validation and test data.

**Parameters**

- **test\_size** (*Union[float, int], optional, default=0.1*) –
- **validation\_size** (*Union[float, int], optional, default=0.1*) –
- **random\_state** (*Union[int, RandomState], optional, default=None*) –
  - If int, random\_state is the seed used by the random number generator;
  - If RandomState instance, random\_state is the random number generator;
  - If None, the random number generator is the RandomState instance used by np.random.

**Returns****train\_data, validation\_data, test\_data** – tuple of ADSData instances**Return type**

tuple

### Examples

```
>>> ds = DatasetFactory.open("data.csv")
>>> train, valid, test = ds.train_validation_test_split()
```

#### `type_of_target()`

Return the target type for the dataset.

##### Returns

**target\_type** – an object of TypedFeature

##### Return type

TypedFeature

### Examples

```
>>> ds = ds.set_target('target_class')
>>> assert(ds.type_of_target() == 'categorical')
```

#### `visualize_transforms()`

Render a representation of the dataset's transform DAG.

#### 18.1.1.8.10 `ads.dataset.exception` module

**exception** `ads.dataset.exception.DatasetError(*args, **kwargs)`

Bases: `BaseException`

Base class for dataset errors.

**exception** `ads.dataset.exception.ValidationError(msg)`

Bases: `DatasetError`

Handles validation errors in dataset.

#### 18.1.1.8.11 `ads.dataset.factory` module

**class** `ads.dataset.factory.CustomFormatReaders`

Bases: `object`

**DEFAULT\_SQL\_ARRAYSIZE** = 50000

**DEFAULT\_SQL\_CHUNKSIZE** = 12007

**DEFAULT\_SQL\_CTU** = False

**DEFAULT\_SQL\_MIL** = 128

**static read\_arff**(*path*, *\*\*kwargs*)

**static read\_avro**(*path: str*, *\*\*kwargs*) → `DataFrame`

**static read\_html**(*path*, *html\_table\_index: Optional[int] = None*, *\*\*kwargs*)

```
static read_json(path: str, **kwargs) → DataFrame
```

```
static read_libsvm(path: str, **kwargs) → DataFrame
```

```
static read_log(path, **kwargs)
```

```
classmethod read_sql(path: str, table: Optional[str] = None, **kwargs) → DataFrame
```

#### Parameters

- **path** – str This is the connection URL that gets passed to sqlalchemy’s create\_engine method
- **table** – str This is either the name of a table to select \* from or a sql query to be run
- **kwargs** –

#### Returns

pd.DataFrame

```
static read_tsv(path: str, **kwargs) → DataFrame
```

```
static read_xml(path: str, **kwargs) → DataFrame
```

Load data from xml file.

#### Parameters

- **path** (str) – Path to XML file
- **storage\_options** (dict, optional) – Storage options passed to Pandas to read the file.

#### Returns

dataframe

#### Return type

pandas.DataFrame

```
class ads.dataset.factory.DatasetFactory
```

Bases: object

```
static download(remote_path, local_path, storage=None, overwrite=False)
```

Download a remote file or directory to local storage.

#### Parameters

- **remote\_path** (str) – Supports protocols like oci, s3, also supports glob expressions
- **local\_path** (str) – Supports glob expressions
- **storage** (dict) – Parameters passed on to the backend remote filesystem class.
- **overwrite** (bool, default False) – If True, the method will overwrite any existing files in the local\_path

## Examples

```
>>> DatasetFactory.download("oci://Bucket/prefix/to/data/*.csv",
...                          "/home/datascience/data/")
```

**static** `from_dataframe(df, target: Optional[str] = None, **kwargs)`

Returns an object of `ADSDatasetWithTarget` or `ADSDataset` given a `pandas.DataFrame`

### Parameters

- **df** (`pandas.DataFrame`) –
- **target** (`str`) –
- **kwargs** (`dict`) – See `DatasetFactory.open()` for supported kwargs

### Returns

**dataset** – according to the type of target

### Return type

an object of `ADSDataset` target is not specified, otherwise an object of `ADSDatasetWithTarget` tagged

## Examples

```
>>> df = pd.DataFrame(data)
>>> ds = from_dataframe(df)
```

**classmethod** `infer_target_type(target, target_series, discover_target_type=True)`

**static** `list_snapshots(snapshot_dir=None, name="", storage_options=None, **kwargs)`

Displays the URIs for dataset snapshots under the given directory path.

### Parameters

- **snapshot\_dir** (`str`) – Return all dataset snapshots created using `ADSDataset.snapshot()` within this directory. The path can contain protocols such as `oci`, `s3`.
- **name** (`str`, *optional*) – The list of snapshots in the directory gets filtered by the name. Accepts glob expressions. default = “`ads_`”
- **storage\_options** (`dict`) – Parameters passed on to the backend filesystem class.

## Example

```
>>> DatasetFactory.list_snapshots(snapshot_dir="oci://my_bucket/snapshots_dir",
...                               name="ads_iris_")
```

Returns a list of all snapshots (recursively) saved to obj storage bucket “`my_bucket`” with prefix “`/snapshots_dir/ads_iris_*`” sorted by time created.

**static** `open(source, target=None, format='infer', reader_fn: Callable = None, name: str = None, description="", npartitions: int = None, type_discovery=True, html_table_index=None, column_names='infer', sample_max_rows=10000, positive_class=None, transformer_pipeline=None, types={}, **kwargs)`



Returns an object of ADSDataset or ADSDatasetWithTarget read from the given path

Deprecated since version 2.6.6: “Deprecated in favor of using Pandas. Pandas supports reading from object storage directly. Check [https://accelerated-data-science.readthedocs.io/en/latest/user\\_guide/loading\\_data/connect.html](https://accelerated-data-science.readthedocs.io/en/latest/user_guide/loading_data/connect.html)”,

#### Parameters

- **source** (*Union[str, pandas.DataFrame, h2o.DataFrame, pyspark.sql.dataframe.DataFrame]*) – If str, URI for the dataset. The dataset could be read from local or network file system, hdfs, s3, gcs and optionally pyspark in pyspark conda env
- **target** (*str, optional*) – Name of the target in dataset. If set an ADSDatasetWithTarget object is returned, otherwise an ADSDataset object is returned which can be used to understand the dataset through visualizations
- **format** (*str, default: infer*) – Format of the dataset. Supported formats: CSV, TSV, Parquet, libsvm, JSON, XLS/XLSX (Excel), HDF5, SQL, XML, Apache server log files (clf, log), ARFF. By default, the format would be inferred from the ending of the dataset file path.
- **reader\_fn** (*Callable, default: None*) – The user may pass in their own custom reader function. It must accept (*path, \*\*kwargs*) and return a pandas DataFrame
- **name** (*str, optional default: ""*) –
- **description** (*str, optional default: ""*) – Text describing the dataset
- **npartitions** (*int, deprecated*) – Number of partitions to split the data By default this is set to the max number of cores supported by the backend compute accelerator
- **type\_discovery** (*bool, default: True*) – If false, the data types of the dataframe are used as such. By default, the dataframe columns are associated with the best suited data types. Associating the features with the discovered datatypes would impact visualizations and model prediction.
- **html\_table\_index** (*int, optional*) – The index of the dataframe table in html content. This is used when the format of dataset is html
- **column\_names** (*'infer', list of str or None, default: 'infer'*) – Supported only for CSV and TSV. List of column names to use. By default, column names are inferred from the first line of the file. If set to None, column names would be auto-generated instead of inferring from file. If the file already contains a column header, specify header=0 to ignore the existing column names.
- **sample\_max\_rows** (*int, default: 10000, use -1 auto calculate sample size, use 0 (zero) for no sampling*) – Sample size of the dataframe to use for visualization and optimization.
- **positive\_class** (*Any, optional*) – Label in target for binary classification problems which should be identified as positive for modeling. By default, the first unique value is considered as the positive label.
- **types** (*dict, optional*) – Dictionary of <feature\_name> : <data\_type> to override the data type of features.
- **transformer\_pipeline** (*datasets.pipeline.TransformerPipeline, optional*) – A pipeline of transformations done outside the sdk and need to be applied at the time of scoring

- **storage\_options** (*dict*, *default: varies by source type*) – Parameters passed on to the backend filesystem class.
- **sep** (*str*) – Delimiting character for parsing the input file.
- **kwargs** (*additional keyword arguments that would be passed to underlying dataframe read API*) – based on the format of the dataset

#### Returns

- **dataset** (*An instance of ADSDataset*)
- (*or*)
- **dataset\_with\_target** (*An instance of ADSDatasetWithTarget*)

#### Examples

```
>>> ds = DatasetFactory.open("/path/to/data.data", format='csv', delimiter=" ",
...                             na_values="n/a", skipinitialspace=True)
```

```
>>> ds = DatasetFactory.open("/path/to/data.csv", target="col_1", prefix="col_",
...                             skiprows=1, encoding="ISO-8859-1")
```

```
>>> ds = DatasetFactory.open("oci://bucket@namespace/path/to/data.tsv",
...                             column_names=["col1", "col2", "col3"], header=0)
```

```
>>> ds = DatasetFactory.open("oci://bucket@namespace/path/to/data.csv",
...                             storage_options={"config": "~/oci/config",
...                             "profile": "USER_2"}, delimiter = ';')
```

```
>>> ds = DatasetFactory.open("/path/to/data.parquet", engine='pyarrow',
...                             types={"col1": "ordinal",
...                                     "col2": "categorical",
...                                     "col3" : "continuous",
...                                     "col4" : "float64"})
```

```
>>> ds = DatasetFactory.open(df, target="class", sample_max_rows=5000,
...                             positive_class="yes")
```

```
>>> ds = DatasetFactory.open("s3://path/to/data.json.gz", format="json",
...                             compression="gzip", orient="records")
```

**static** `open_to_pandas`(*source: str, format: Optional[str] = None, reader\_fn: Optional[Callable] = None, \*\*kwargs*) → `DataFrame`

**static** `set_default_storage`(*snapshots\_dir=None, storage\_options=None*)

Set default storage directory and options.

Both `snapshots_dir` and `storage_options` can be overridden at the API scope.

#### Parameters

- **snapshots\_dir** (*str*) – Path for the snapshots directory. Can contain protocols such as `oci`, `s3`

- **storage\_options** (*dict*, *optional*) – Parameters passed on to the backend filesystem class.

**static upload**(*local\_file\_or\_dir*, *remote\_file\_or\_dir*, *storage\_options=None*)

Upload local file or directory to remote storage

#### Parameters

- **local\_file\_or\_dir** (*str*) – Supports glob expressions
- **remote\_file\_or\_dir** (*str*) – Supports protocols like oci, s3, also supports glob expressions
- **storage\_options** (*dict*) – Parameters passed on to the backend remote filesystem class.

`ads.dataset.factory.get_format_reader(path: ElaboratedPath, **kwargs) → Callable`

`ads.dataset.factory.load_dataset(path: ElaboratedPath, reader_fn: Callable, **kwargs) → DataFrame`

#### 18.1.1.8.12 ads.dataset.feature\_engineering\_transformer module

**class** `ads.dataset.feature_engineering_transformer.FeatureEngineeringTransformer`(*feature\_metadata=None*)

Bases: `TransformerMixin`

**fit**(*X*, *y=None*)

**fit\_transform**(*X*, *y=None*, *\*\*fit\_params*)

Fit to data, then transform it.

Fits transformer to *X* and *y* with optional parameters *fit\_params* and returns a transformed version of *X*.

#### Parameters

- **X** (*array-like of shape (n\_samples, n\_features)*) – Input samples.
- **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs), default=None*) – Target values (None for unsupervised transformations).
- **\*\*fit\_params** (*dict*) – Additional fit parameters.

#### Returns

**X\_new** – Transformed array.

#### Return type

ndarray array of shape (n\_samples, n\_features\_new)

**transform**(*df*, *progress=<ads.dataset.progress.DummyProgressBar object>*, *fit\_transform=False*)

#### 18.1.1.8.13 ads.dataset.feature\_selection module

**class** `ads.dataset.feature_selection.FeatureImportance`(*ds*, *score\_func=None*, *n=None*)

Bases: `object`

**show\_in\_notebook**(*fig\_size=(10, 10)*)

Shows selected features in the notebook with matplotlib.

#### 18.1.1.8.14 ads.dataset.forecasting\_dataset module

```
class ads.dataset.forecasting_dataset.ForecastingDataset(df, sampled_df, target, target_type, shape,
                                                         **kwargs)
```

Bases: *ADSDatasetWithTarget*

```
select_best_features(score_func=None, k=12)
```

Not yet implemented

#### 18.1.1.8.15 ads.dataset.helper module

```
class ads.dataset.helper.DatasetDefaults
```

Bases: object

```
sampling_confidence_interval = 1.0
```

```
sampling_confidence_level = 95
```

```
exception ads.dataset.helper.DatasetLoadException(exc_msg)
```

Bases: *BaseException*

```
class ads.dataset.helper.ElaboratedPath(source: Union[str, List[str]], format: Optional[str] = None,
                                         name: Optional[str] = None, **kwargs)
```

Bases: object

The Elaborated Path class unifies all of the operations and information related to a path or pathlist. Whether the user wants to An Elaborated path can accept any of the following as a valid source: \* A single path \* A glob pattern path \* A directory \* A list of paths (Note: all of these paths must be from the same filesystem AND have the same format) \* A sqlalchemy connection url

##### Parameters

- **source** –
- **format** –
- **kwargs** –

By the end of this method, this class needs to have paths, format, and name ready

**property format: str**

**property name: str**

**property num\_paths: int**

This method will return the number of paths found with the associated original glob, folder, or path. If this returns 0, :return:

**property paths: List[str]**

a list of str Each element will be a valid path

##### Type

return

```
ads.dataset.helper.calculate_sample_size(population_size, min_size_to_sample, confidence_level=95,
                                         confidence_interval=1.0)
```

**Find sample size for a population using Cochran's Sample Size Formula.**

With default values for `confidence_level` (percentage, default: 95%) and `confidence_interval` (margin of error, percentage, default: 1%)

SUPPORTED CONFIDENCE LEVELS: 50%, 68%, 90%, 95%, and 99% *ONLY* - this is because the Z-score is table based, and I'm only providing Z for common confidence levels.

```
ads.dataset.helper.concatenate(X, y)
```

```
ads.dataset.helper.convert_columns(df, feature_metadata=None, dtypes=None)
```

```
ads.dataset.helper.convert_to_html(plot)
```

```
ads.dataset.helper.deprecate_default_value(var, old_value, new_value, warning_msg, warning_type)
```

```
ads.dataset.helper.deprecate_variable(old_var, new_var, warning_msg, warning_type)
```

```
ads.dataset.helper.down_sample(df, target)
```

Fixes imbalanced dataset by down-sampling

**Parameters**

- **df** (*pandas.DataFrame*) –
- **target** (*name of the target column in df*) –

**Returns**

**downsampled\_df**

**Return type**

*pandas.DataFrame*

```
ads.dataset.helper.fix_column_names(X)
```

```
ads.dataset.helper.generate_sample(df: DataFrame, n: int, confidence_level: int = 95, confidence_interval: float = 1.0, **kwargs)
```

```
ads.dataset.helper.get_dtype(feature_type, dtype)
```

```
ads.dataset.helper.get_feature_type(name, series)
```

```
ads.dataset.helper.get_fill_val(feature_types, column, action, constant='constant')
```

```
ads.dataset.helper.is_text_data(df, target=None)
```

```
ads.dataset.helper.map_types(types)
```

```
ads.dataset.helper.parse_apache_log_datetime(x)
```

**Parses datetime with timezone formatted as:**

*[day/month/year:hour:minute:second zone]*

Source: <https://mmas.github.io/read-apache-access-log-pandas> .. rubric:: Example

```
>>> parse_datetime('13/Nov/2015:11:45:42 +0000')
datetime.datetime(2015, 11, 3, 11, 45, 4, tzinfo=<UTC>)
```

Due to problems parsing the timezone (%z) with `datetime.strptime`, the timezone will be obtained using the `pytz` library.

`ads.dataset.helper.parse_apache_log_str(x)`

Returns the string delimited by two characters.

Source: [https://mmas.github.io/read-apache-access-log-pandas .. rubric::](https://mmas.github.io/read-apache-access-log-pandas..rubric::) Example

```
>>> parse_str('[my string]') 'my string'
```

`ads.dataset.helper.rename_duplicate_cols(original_cols)`

`ads.dataset.helper.up_sample(df, target, sampler='default', feature_types=None)`

Fixes imbalanced dataset by up-sampling

#### Parameters

- **df** (*Union[pandas.DataFrame, dask.dataframe.core.DataFrame]*) –
- **target** (*name of the target column in df*) –
- **sampler** (*Should implement fit\_resample(X,y) method*) –
- **fillna** (*a dictionary contains the column name as well as the fill value,*) – only needed when the column has missing values

#### Returns

**upsampled\_df**

#### Return type

*Union[pandas.DataFrame, dask.dataframe.core.DataFrame]*

`ads.dataset.helper.visualize_transformation(transformer_pipeline, text=None)`

`ads.dataset.helper.write_parquet(path, data, engine='fastparquet', metadata_dict=None, compression=None, storage_options=None)`

Uses fast parquet to write dask dataframe and custom metadata in parquet format

#### Parameters

- **path** (*str*) – Path to write to
- **data** (*pandas.DataFrame*) –
- **engine** (*string*) – “auto” by default
- **metadata\_dict** (*Deprecated, will not pass through*) –
- **compression** (*{{'snappy', 'gzip', 'brotli', None}}, default 'snappy'*) – Name of the compression to use
- **storage\_options** (*dict, optional*) – storage arguments required to read the path

#### Returns

**str**

#### Return type

the file path the parquet was written to

#### 18.1.1.8.16 `ads.dataset.label_encoder` module

**class** `ads.dataset.label_encoder.DataFrameLabelEncoder`

Bases: `TransformerMixin`

Label encoder for `pandas.dataframe`. `dask.dataframe.core.DataFrame`

**fit**(*X*)

Fits a `DataFrameLabelEncoder`.

**transform**(*X*)

Transforms a dataset using the `DataFrameLabelEncoder`.

#### 18.1.1.8.17 `ads.dataset.pipeline` module

**class** `ads.dataset.pipeline.TransformerPipeline`(*steps*)

Bases: `Pipeline`

**add**(*transformer*)

Add transformer to data transformation pipeline

**Parameters**

**transformer** (`Union[TransformerMixin, tuple(str, TransformerMixin)]`) – if tuple, (name, transformer implementing transform)

**steps**: `List[Any]`

**visualize**()

#### 18.1.1.8.18 `ads.dataset.plot` module

**class** `ads.dataset.plot.Plotting`(*df*, *feature\_types*, *x*, *y=None*, *plot\_type='infer'*, *yscale=None*)

Bases: `object`

**select\_best\_plot**()

Returns the best plot for a given dataset

**show\_in\_notebook**(*\*\*kwargs*)

Visualizes the dataset by plotting the distribution of a feature or relationship between two features.

**Parameters**

- **figsize** (*tuple*) – defines the size of the fig
- -----

#### 18.1.1.8.19 ads.dataset.progress module

```
class ads.dataset.progress.DummyProgressBar(*args, **kwargs)
    Bases: ProgressBar
    update(*args, **kwargs)
        Updates the progress bar

class ads.dataset.progress.IpythonProgressBar(max_progress=100, description='Running',
                                              verbose=False)
    Bases: ProgressBar
    update(description=None)
        Updates the progress bar

class ads.dataset.progress.ProgressBar
    Bases: object
    abstract update(description)

class ads.dataset.progress.TqdmProgressBar(max_progress=100, description='Running', verbose=False)
    Bases: ProgressBar
    update(description=None)
        Updates the progress bar
```

#### 18.1.1.8.20 ads.dataset.recommendation module

```
class ads.dataset.recommendation.Recommendation(ds, recommendation_transformer)
    Bases: object
    recommendation_type_labels = ['Constant Columns', 'Potential Primary Key Columns',
                                  'Imputation', 'Multicollinear Columns', 'Identify positive label for target', 'Fix
                                  imbalance in dataset']
    recommendation_types = ['constant_column', 'primary_key', 'imputation',
                             'strong_correlation', 'positive_class', 'fix_imbalance']
    show_in_notebook()
```

#### 18.1.1.8.21 ads.dataset.recommendation\_transformer module

```
class ads.dataset.recommendation_transformer.RecommendationTransformer(feature_metadata=None,
                                                                           correlation=None,
                                                                           target=None,
                                                                           is_balanced=False,
                                                                           target_type=None,
                                                                           feature_ranking=None,
                                                                           len=0,
                                                                           fix_imbalance=True,
                                                                           auto_transform=True,
                                                                           correla-
                                                                           tion_threshold=0.7)
```



Bases: `TransformerMixin`

**fit**(*X*)

**fit\_transform**(*X*, *y=None*, *\*\*fit\_params*)

Fit to data, then transform it.

Fits transformer to *X* and *y* with optional parameters *fit\_params* and returns a transformed version of *X*.

#### Parameters

- **X** (array-like of shape (*n\_samples*, *n\_features*)) – Input samples.
- **y** (array-like of shape (*n\_samples*,) or (*n\_samples*, *n\_outputs*), *default=None*) – Target values (None for unsupervised transformations).
- **\*\*fit\_params** (*dict*) – Additional fit parameters.

#### Returns

**X\_new** – Transformed array.

#### Return type

ndarray array of shape (*n\_samples*, *n\_features\_new*)

**transform**(*X*, *progress=<ads.dataset.progress.DummyProgressBar object>*, *fit\_transform=False*, *update\_transformer\_log=False*)

**transformer\_log**(*action*)

local wrapper to both log and record in the *actions\_performed* array

### 18.1.1.8.22 ads.dataset.regression\_dataset module

**class** `ads.dataset.regression_dataset.RegressionDataset`(*df*, *sampled\_df*, *target*, *target\_type*, *shape*, *\*\*kwargs*)

Bases: `ADSDatasetWithTarget`

### 18.1.1.8.23 ads.dataset.sampled\_dataset module

**class** `ads.dataset.sampled_dataset.PandasDataset`(*sampled\_df*, *type\_discovery=True*, *types={}*, *metadata=None*, *progress=<ads.dataset.progress.DummyProgressBar object>*)

Bases: `object`

This class provides APIs that can work on a sampled dataset.

**plot**(*x*, *y=None*, *plot\_type='infer'*, *yscale=None*, *verbose=True*, *sample\_size=0*)

Supports plotting feature distribution, and relationship between features.

#### Parameters

- **x** (*str*) – The name of the feature to plot
- **y** (*str*, *optional*) – Name of the feature to plot against x
- **plot\_type** (*str*, *default: infer*) – Override the inferred plot type for certain combinations of the data types of *x* and *y*. By default, the best plot type is inferred based on *x* and *y* data types. Valid values:

- `box_plot` - discrete feature vs continuous feature. Draw a box plot to show distributions with respect to categories,
- `scatter` - continuous feature vs continuous feature. Draw a scatter plot with possibility of several semantic groupings.
- **`yscale`** (*str*, *optional*) – One of {"linear", "log", "symlog", "logit"}. The y axis scale type to apply. Can be used when either x or y is an ordinal feature.
- **`verbose`** (*bool*, *default True*) – Displays Note/Tips if True

**`plot_gis_scatter`** (*lon='longitude', lat='latitude', ax=None*)

Supports plotting Choropleth maps

**Parameters**

- **`df`** (*pandas dataframe*) – The dataframe to plot
- **`x`** (*str*) – The name of the feature to plot, usually the longitude
- **`y`** (*str*) – The name of the feature to plot, usually the latitude

**`summary`** (*feature\_name=None*)

Display list of features & their datatypes. Shows the column name and the feature's meta\_data if given a specific feature name.

**Parameters**

**`date_col`** (*str*) – The name of the feature

**Returns**

a dictionary that contains requested information

**Return type**

dict

**`timeseries`** (*date\_col*)

Supports any plotting operations where x=datetime.

**Parameters**

**`date_col`** (*str*) – The name of the feature to plot

**Returns**

a plotting object that contains a date column and dataframe

**Return type**

func

#### 18.1.1.8.24 `ads.dataset.target` module

**`class ads.dataset.target.TargetVariable`** (*sampled\_ds, target, target\_type*)

Bases: object

This class provides target specific APIs.

**`is_balanced`** (*skewness\_threshold=0.5, class\_imbalance\_threshold=0.5*)

Returns True if the target is balanced, False otherwise.

**Returns**

**`is_balanced`**

**Return type**

bool

**show\_in\_notebook**(*feature\_names=None*)

Plot target distribution or target versus feature relation.

**Parameters**

**feature\_names** (*list, Optional*) – Plot target against a list of features. Display target distribution if *feature\_names* is not provided.

#### 18.1.1.8.25 `ads.dataset.timeseries` module

**class** `ads.dataset.timeseries.Timeseries`(*col\_name, df, date\_range=None, min=None, max=None*)

Bases: `object`

**plot**(\*\**kwargs*)

#### 18.1.1.8.26 Module contents

### 18.1.1.9 `ads.evaluations` package

#### 18.1.1.9.1 Submodules

#### 18.1.1.9.2 `ads.evaluations.evaluation_plot` module

**class** `ads.evaluations.evaluation_plot.EvaluationPlot`

Bases: `object`

`EvaluationPlot` holds data and methods for plots and it used to output them

**baseline**(*bool*)

whether to plot the null model or zero information model

**baseline\_kwargs**(*dict*)

keyword arguments for the baseline plot

**color\_wheel**(*dict*)

color information used by the plot

**font\_sz**(*dict*)

dictionary of plot methods

**perfect**(*bool*)

determines whether a “perfect” classifier curve is displayed

**perfect\_kwargs**(*dict*)

parameters for the perfect classifier for precision/recall curves

**prob\_type**(*str*)

model type, i.e. classification or regression

**get\_legend\_labels**(*legend\_labels*)

Renders the legend labels on the plot

**plot**(*evaluation, plots, num\_classes, perfect, baseline, legend\_labels*)

Generates the evaluation plot

```
baseline = None

baseline_kwargs = {'c': '.2', 'ls': '--'}

color_wheel = ['teal', 'blueviolet', 'forestgreen', 'peru', 'y', 'dodgerblue', 'r']

double_overlay_plots = ['pr_and_roc_curve', 'lift_and_gain_chart']

font_sz = {'l': 14, 'm': 12, 's': 10, 'xl': 16, 'xs': 8}
```

**classmethod** `get_legend_labels(legend_labels)`

Gets the legend labels, resolves any conflicts such as length, and renders the labels for the plot

**Parameters**

(**dict**) (*legend\_labels*) – key/value dictionary containing legend label data

**Return type**

Nothing

### Examples

```
EvaluationPlot.get_legend_labels({'class_0': 'green', 'class_1': 'yellow', 'class_2': 'red'})
```

```
perfect = None
```

```
perfect_kwargs = {'color': 'gold', 'label': 'Perfect Classifier', 'ls': '--'}
```

**classmethod** `plot(evaluation, plots, num_classes, perfect=False, baseline=True, legend_labels=None)`

Generates the evaluation plot

**Parameters**

- (**DataFrame**) (*evaluation*) – DataFrame with models as columns and metrics as rows.
- (**str**) (*plots*) – The plot type based on class attribute *prob\_type*.
- (**int**) (*num\_classes*) – The number of classes for the model.
- (**bool** (*baseline*)) – Whether to display the curve of a perfect classifier. Default value is *False*.
- (**optional**) – Whether to display the curve of a perfect classifier. Default value is *False*.
- (**bool** – Whether to display the curve of the baseline, featureless model. Default value is *True*.
- (**optional**) – Whether to display the curve of the baseline, featureless model. Default value is *True*.
- (**dict** (*legend\_labels*)) – Legend labels dictionary. Default value is *None*. If *legend\_labels* not specified class names will be used for plots.
- (**optional**) – Legend labels dictionary. Default value is *None*. If *legend\_labels* not specified class names will be used for plots.

**Return type**

Nothing

```
prob_type = None
```

```
single_overlay_plots = ['lift_chart', 'gain_chart', 'roc_curve', 'pr_curve']
```

### 18.1.1.9.3 ads.evaluations.evaluator module

**class** ads.evaluations.evaluator.**ADSEvaluator**(*test\_data, models, training\_data=None, positive\_class=None, legend\_labels=None, show\_full\_name=False*)

Bases: object

ADS Evaluator class. This class holds field and methods for creating and using ADS evaluator objects.

#### **evaluations**

list of evaluations.

#### **Type**

list[DataFrame]

#### **is\_classifier**

Whether the model has a non-empty *classes\_* attribute indicating the presence of class labels.

#### **Type**

bool

#### **legend\_labels**

List of legend labels. Defaults to *None*.

#### **Type**

dict

#### **metrics\_to\_show**

Names of metrics to show.

#### **Type**

list[str]

#### **models**

The object built using *ADSModel.from\_estimator()*.

#### **Type**

list[[ads.common.model.ADSModel](#)]

#### **positive\_class**

The class to report metrics for binary dataset, assumed to be true.

#### **Type**

str or int

#### **show\_full\_name**

Whether to show the name of the evaluator in relevant contexts.

#### **Type**

bool

#### **test\_data**

Test data to evaluate model on.

#### **Type**

[ads.common.data.ADSDData](#)

#### **training\_data**

Training data to evaluate model.

**Type***ads.common.data.ADSDData***Positive\_Class\_names**

Class attribute listing the ways to represent positive classes

**Type**

list

**add\_metrics**(*func, names*)

Adds the listed metics to the evaluator it is called on

**del\_metrics**(*names*)

Removes listed metrics from the evaluator object it is called on

**add\_models**(*models, show\_full\_name*)

Adds the listed models to the evaluator object

**del\_models**(*names*)

Removes the listed models from the evaluator object

**show\_in\_notebook**(*plots, use\_training\_data, perfect, baseline, legend\_labels*)

Visualize evaluation plots in the notebook

**calculate\_cost**(*tn\_weight, fp\_weight, fn\_weight, tp\_weight, use\_training\_data*)

Returns a cost associated with the input weights

Creates an ads evaluator object.

**Parameters**

- **test\_data** (*ads.common.data.ADSDData instance*) – Test data to evaluate model on. The object can be built using *ADSDData.build()*.
- **models** (*list[ads.common.model.ADSModel]*) – The object can be built using *ADSModel.from\_estimator()*. Maximum length of the list is 3
- **training\_data** (*ads.common.data.ADSDData instance, optional*) – Training data to evaluate model on and compare metrics against test data. The object can be built using *ADSDData.build()*
- **positive\_class** (*str or int, optional*) – The class to report metrics for binary dataset. If the target classes is True or False, positive\_class will be set to True by default. If the dataset is multiclass or multilabel, this will be ignored.
- **legend\_labels** (*dict, optional*) – List of legend labels. Defaults to *None*. If legend\_labels not specified class names will be used for plots.
- **show\_full\_name** (*bool, optional*) – Show the name of the evaluator object. Defaults to *False*.

## Examples

```
>>> train, test = ds.train_test_split()
>>> model1 = MyModelClass1.train(train)
>>> model2 = MyModelClass2.train(train)
>>> evaluator = ADSEvaluator(test, [model1, model2])
```

```
>>> legend_labels={'class_0': 'one', 'class_1': 'two', 'class_2': 'three'}
>>> multi_evaluator = ADSEvaluator(test, models=[model1, model2],
...                               legend_labels=legend_labels)
```

**class EvaluationMetrics**(*ev\_test, ev\_train, use\_training=False, less\_is\_more=None, precision=4*)

Bases: object

Class holding evaluation metrics.

**ev\_test**

evaluation test metrics

**Type**

list

**ev\_train**

evaluation training metrics

**Type**

list

**use\_training**

use training data

**Type**

bool

**less\_is\_more**

metrics list

**Type**

list

**show\_in\_notebook()**

Shows visualization metrics as a color coded table

```
DEFAULT_LABELS_MAP = {'accuracy': 'Accuracy', 'auc': 'ROC AUC', 'f1': 'F1',
                      'hamming_loss': 'Hamming distance', 'kappa_score_': "Cohen's kappa coefficient",
                      'precision': 'Precision', 'recall': 'Recall'}
```

**property precision**

```
show_in_notebook(labels={'accuracy': 'Accuracy', 'auc': 'ROC AUC', 'f1': 'F1', 'hamming_loss':
                        'Hamming distance', 'kappa_score_': "Cohen's kappa coefficient", 'precision':
                        'Precision', 'recall': 'Recall'})
```

Visualizes evaluation metrics as a color coded table.

**Parameters**

**labels** (*dictionary*) – map printing specific labels for metrics display

**Return type**

Nothing

**Positive\_Class\_Names** = ['yes', 'y', 't', 'true', '1']

**add\_metrics**(*funcs, names*)

Adds the listed metrics to the evaluator object it is called on.

**Parameters**

- **funcs** (*list*) – The list of metrics to be added. This function will be provided *y\_true* and *y\_pred*, the true and predicted values for each model.
- **names** (*list[str]*) – The list of metric names corresponding to the functions.

**Return type**

Nothing

**Examples**

```
>>> def f1(y_true, y_pred):
...     return np.max(y_true - y_pred)
>>> evaluator = ADSEvaluator(test, [model1, model2])
>>> evaluator.add_metrics([f1], ['Max Residual'])
>>> evaluator.metrics
Output table will include the desired metric
```

**add\_models**(*models, show\_full\_name=False*)

Adds the listed models to the evaluator object it is called on.

**Parameters**

- **models** (*list[ADSMoDel]*) – The list of models to be added
- **show\_full\_name** (*bool, optional*) – Whether to show the full model name. Defaults to False. **\*\* NOT USED \*\***

**Return type**

Nothing

**Examples**

```
>>> evaluator = ADSEvaluator(test, [model1, model2])
>>> evaluator.add_models("model3"])
```

**calculate\_cost**(*tn\_weight, fp\_weight, fn\_weight, tp\_weight, use\_training\_data=False*)

Returns a cost associated with the input weights.

**Parameters**

- **tn\_weight** (*int, float*) – The weight to assign true negatives in calculating the cost
- **fp\_weight** (*int, float*) – The weight to assign false positives in calculating the cost
- **fn\_weight** (*int, float*) – The weight to assign false negatives in calculating the cost
- **tp\_weight** (*int, float*) – The weight to assign true positives in calculating the cost
- **use\_training\_data** (*bool, optional*) – Use training data to pull the metrics. Defaults to False

**Returns**

DataFrame with the cost calculated for each model



**Return type**  
pandas.DataFrame

### Examples

```
>>> evaluator = ADSEvaluator(test, [model1, model2])
>>> costs_table = evaluator.calculate_cost(0, 10, 1000, 0)
```

#### **del\_metrics**(names)

Removes the listed metrics from the evaluator object it is called on.

##### **Parameters**

**names** (*list[str]*) – The list of names of metrics to be deleted. Names can be found by calling *evaluator.test\_evaluations.index*.

##### **Returns**

*None*

##### **Return type**

*None*

### Examples

```
>>> evaluator = ADSEvaluator(test, [model1, model2])
>>> evaluator.del_metrics(['mse'])
>>> evaluator.metrics
Output table will exclude the desired metric
```

#### **del\_models**(names)

Removes the listed models from the evaluator object it is called on.

##### **Parameters**

**names** (*list[str]*) – the list of models to be delete. Names are the model names by default, and assigned internally when conflicts exist. Actual names can be found using *evaluator.test\_evaluations.columns*

##### **Return type**

*Nothing*

### Examples

```
>>> model3.rename("model3")
>>> evaluator = ADSEvaluator(test, [model1, model2, model3])
>>> evaluator.del_models([model3])
```

#### **property metrics**

Returns evaluation metrics

##### **Returns**

HTML representation of a table comparing relevant metrics.

##### **Return type**

metrics

## Examples

```
>>> evaluator = ADSEvaluator(test, [model1, model2])
>>> evaluator.metrics
Outputs table displaying metrics.
```

### property `raw_metrics`

Returns the raw metric numbers

#### Parameters

- **metrics** (*list, optional*) – Request metrics to pull. Defaults to all.
- **use\_training\_data** (*bool, optional*) – Use training data to pull metrics. Defaults to `False`

#### Returns

The requested raw metrics for each model. If *metrics* is *None* return all.

#### Return type

dict

## Examples

```
>>> evaluator = ADSEvaluator(test, [model1, model2])
>>> raw_metrics_dictionary = evaluator.raw_metrics()
```

**show\_in\_notebook** (*plots=None, use\_training\_data=False, perfect=False, baseline=True, legend\_labels=None*)

Visualize evaluation plots.

#### Parameters

- **plots** (*list, optional*) – Filter the plots that are displayed. Defaults to *None*. The name of the plots are as below:
  - regression - residuals\_qq, residuals\_vs\_fitted
  - binary classification - normalized\_confusion\_matrix, roc\_curve, pr\_curve
  - multi class classification - normalized\_confusion\_matrix, precision\_by\_label, recall\_by\_label, f1\_by\_label
- **use\_training\_data** (*bool, optional*) – Use training data to generate plots. Defaults to *False*. By default, this method uses test data to generate plots
- **legend\_labels** (*dict, optional*) – Rename legend labels, that used for multi class classification plots. Defaults to *None*. *legend\_labels* dict keys are the same as class names. *legend\_labels* dict values are strings. If *legend\_labels* not specified class names will be used for plots.

#### Returns

Nothing. Outputs several evaluation plots as specified by *plots*.

#### Return type

*None*

## Examples

```
>>> evaluator = ADSEvaluator(test, [model1, model2])
>>> evaluator.show_in_notebook()
```

```
>>> legend_labels={'class_0': 'green', 'class_1': 'yellow', 'class_2': 'red'}
>>> multi_evaluator = ADSEvaluator(test, [model1, model2],
...     legend_labels=legend_labels)
>>> multi_evaluator.show_in_notebook(plots=["normalized_confusion_matrix",
...     "precision_by_label", "recall_by_label", "f1_by_label"])
```

### 18.1.1.9.4 ads.evaluations.statistical\_metrics module

**class** ads.evaluations.statistical\_metrics.**ModelEvaluator**(*y\_true, y\_pred, model\_name,*  
*classes=None, positive\_class=None,*  
*y\_score=None*)

Bases: object

ModelEvaluator takes in the true and predicted values and returns a pandas dataframe

**y\_true**

**Type**

array-like object holding the true values for the model

**y\_pred**

**Type**

array-like object holding the predicted values for the model

**model\_name**(*str*)

**Type**

the name of the model

**classes**(*list*)

**Type**

list of target classes

**positive\_class**(*str*)

**Type**

label for positive outcome from model

**y\_score**

**Type**

array-like object holding the scores for true values for the model

**metrics**(*dict*)

**Type**

dictionary object holding model data

**get\_metrics**()

Gets the metrics information in a dataframe based on the number of classes

**safe\_metrics\_call**(*scoring\_functions*, \\*args)

Applies sklearn scoring functions to parameters in args

**get\_metrics**()

Gets the metrics information in a dataframe based on the number of classes

**Parameters**

**self** ((*ModelEvaluator* instance)) – The *ModelEvaluator* instance with the metrics.

**Returns**

Pandas dataframe containing the metrics

**Return type**

pandas.DataFrame

**safe\_metrics\_call**(*scoring\_functions*, \*args)

Applies the sklearn function in *scoring\_functions* to parameters in *args*.

**Parameters**

- **scoring\_functions** ((*dict*)) – Scoring functions dictionary
- **args** ((*keyword arguments*)) – Arguments passed to the sklearn function from metrics

**Returns**

Nothing

**Raises**

**Exception** – If an error is encountered applying the sklearn function fn to arguments.

#### 18.1.1.9.5 Module contents

#### 18.1.1.10 ads.explanations package

##### 18.1.1.10.1 Submodules

##### 18.1.1.10.2 ads.explanations.base\_explainer module

##### 18.1.1.10.3 ads.explanations.explainer module

##### 18.1.1.10.4 ads.explanations.mlx\_global\_explainer module

##### 18.1.1.10.5 ads.explanations.mlx\_interface module

##### 18.1.1.10.6 ads.explanations.mlx\_local\_explainer module

##### 18.1.1.10.7 ads.explanations.mlx\_whatif\_explainer module

##### 18.1.1.10.8 Module contents

#### 18.1.1.11 ads.feature\_engineering package

##### 18.1.1.11.1 Submodules

##### 18.1.1.11.2 ads.feature\_engineering.exceptions module

**exception** ads.feature\_engineering.exceptions.InvalidFeatureType(*tname: str*)

Bases: TypeError

**exception** ads.feature\_engineering.exceptions.NameAlreadyRegistered(*name: str*)

Bases: NameError

**exception** ads.feature\_engineering.exceptions.TypeAlreadyAdded(*tname: str*)

Bases: TypeError

**exception** ads.feature\_engineering.exceptions.TypeAlreadyRegistered(*tname: str*)

Bases: TypeError

**exception** ads.feature\_engineering.exceptions.TypeNotFound(*tname: str*)

Bases: TypeError

**exception** ads.feature\_engineering.exceptions.WarningAlreadyExists(*name: str*)

Bases: ValueError

**exception** ads.feature\_engineering.exceptions.WarningNotFound(*name: str*)

Bases: ValueError

### 18.1.1.11.3 ads.feature\_engineering.feature\_type\_manager module

The module that helps to manage feature types. Provides functionalities to register, unregister, list feature types.

#### Classes

##### FeatureTypeManager

Feature Types Manager class that manages feature types.

#### Examples

```
>>> from ads.feature_engineering.feature_type.base import FeatureType
>>> class NewType(FeatureType):
...     description="My personal type."
...     pass
>>> FeatureTypeManager.feature_type_register(NewType)
>>> FeatureTypeManager.feature_type_registered()
```

	Name	Feature Type	Description
0	Continuous	continuous	Type representing continuous values.
1	DateTime	date_time	Type representing date and/or time.
2	Category	category	Type representing discrete unordered values.
3	Ordinal	ordinal	Type representing ordered values.
4	NewType	new_type	My personal type.

```
>>> FeatureTypeManager.warning_registered()
```

	Feature Type	Warning	Handler
0	continuous	zeros	zeros_handler
1	continuous	high_cardinality	high_cardinality_handler

```
>>> FeatureTypeManager.validator_registered()
```

	Feature Type	Validator	Condition	Handler
0	phone_number	is_phone_number	()	default_handler
1	phone_number	is_phone_number	{'country_code': '+7'}	specific_country_handler
2	credit_card	is_credit_card	()	default_handler

```
>>> FeatureTypeManager.feature_type_unregister(NewType)
>>> FeatureTypeManager.feature_type_reset()
>>> FeatureTypeManager.feature_type_object('continuous')
Continuous
```

```
class ads.feature_engineering.feature_type_manager.FeatureTypeManager
```

Bases: object

Feature Types Manager class that manages feature types.

Provides functionalities to register, unregister, list feature types.

**feature\_type\_object**(cls, feature\_type: Union[FeatureType, str]) → FeatureType

Gets a feature type by class object or name.

**feature\_type\_register**(cls, feature\_type\_cls: FeatureType) → None

Registers a feature type.

**feature\_type\_unregister**(cls, feature\_type\_cls: Union[FeatureType, str]) → None

Unregisters a feature type.

**feature\_type\_reset**(cls) → None

Resets feature types to be default.

**feature\_type\_registered**(cls) → pd.DataFrame

Lists all registered feature types as a DataFrame.

**warning\_registered**(cls) → pd.DataFrame

Lists registered warnings for all registered feature types.

**validator\_registered**(cls) → pd.DataFrame

Lists registered validators for all registered feature types.

Examples

```
>>> from ads.feature_engineering.feature_type.base import FeatureType
>>> class NewType(FeatureType):
...     pass
>>> FeatureTypeManager.register_feature_type(NewType)
>>> FeatureTypeManager.feature_type_registered()
```

	Name	Feature Type	Description
0	Continuous	continuous	Type representing continuous values.
1	DateTime	date_time	Type representing date and/or time.
2	Category	category	Type representing discrete unordered values.
3	Ordinal	ordinal	Type representing ordered values.

```
>>> FeatureTypeManager.warning_registered()
```

	Feature Type	Warning	Handler
0	continuous	zeros	zeros_handler
1	continuous	high_cardinality	high_cardinality_handler

```
>>> FeatureTypeManager.validator_registered()
```

	Feature Type	Validator	Condition	Handler
0	phone_number	is_phone_number	()	default_handler
1	phone_number	is_phone_number	{'country_code': '+7'}	specific_country_handler

(continues on next page)

(continued from previous page)

2	credit_card	is_credit_card	()	default_
	↪ handler			

```
>>> FeatureTypeManager.feature_type_unregister(NewType)
>>> FeatureTypeManager.feature_type_reset()
>>> FeatureTypeManager.feature_type_object('continuous')
Continuous
```

**classmethod** `feature_type_object(feature_type: Union[FeatureType, str]) → FeatureType`

Gets a feature type by class object or name.

**Parameters**

**feature\_type** (*Union[FeatureType, str]*) – The FeatureType subclass or a str indicating feature type.

**Returns**

Found feature type.

**Return type**

*FeatureType*

**Raises**

- **TypeNotFound** – If provided feature type not registered.
- **TypeError** – If provided feature type not a subclass of FeatureType.

**classmethod** `feature_type_register(feature_type_cls: FeatureType) → None`

Registers new feature type.

**Parameters**

**feature\_type** (*FeatureType*) – Subclass of FeatureType to be registered.

**Returns**

Nothing.

**Return type**

None

**Raises**

- **TypeError** – Type is not a subclass of FeatureType.
- **TypeError** – Type has already been registered.
- **NameError** – Name has already been used.

**classmethod** `feature_type_registered() → DataFrame`

Lists all registered feature types as a DataFrame.

**Returns**

The list of feature types in a DataFrame format.

**Return type**

pd.DataFrame

**classmethod** `feature_type_reset() → None`

Resets feature types to be default.

**Returns**

Nothing.



**Return type**  
None

**classmethod `feature_type_unregister`**(*feature\_type*: Union[FeatureType, str]) → None  
Unregisters a feature type.

**Parameters**  
**`feature_type`** ((FeatureType | str)) – The FeatureType subclass or a str indicating feature type.

**Returns**  
Nothing.

**Return type**  
None

**Raises**  
**TypeError** – In attempt to unregister a default feature type.

**classmethod `is_type_registered`**(*feature\_type*: Union[FeatureType, str]) → bool  
Checks if provided feature type registered in the system.

**Parameters**  
**`feature_type`** (Union[FeatureType, str]) – The FeatureType subclass or a str indicating feature type.

**Returns**  
True if provided feature type registered, False otherwise.

**Return type**  
bool

**classmethod `validator_registered`**() → DataFrame  
Lists registered validators for registered feature types.

**Returns**  
The list of registered validators for registered feature types in a DataFrame format.

**Return type**  
pd.DataFrame

Examples

>>> FeatureTypeManager.validator_registered()				
	Feature Type	Validator	Condition	
	↪ Handler			
-----				
↪				
0	phone_number	is_phone_number	()	
↪	default_handler			
1	phone_number	is_phone_number	{'country_code': '+7'}	specific_
↪	country_handler			
2	credit_card	is_credit_card	()	
↪	default_handler			

**classmethod `warning_registered`**() → DataFrame  
Lists registered warnings for all registered feature types.

**Returns**

The list of registered warnings for registered feature types in a DataFrame format.

**Return type**

pd.DataFrame

**Examples**

```
>>> FeatureTypeManager.warning_registered()
Feature Type      Warning      Handler
-----
0    continuous      zeros      zeros_handler
1    continuous  high_cardinality  high_cardinality_handler
```

**18.1.1.11.4 ads.feature\_engineering.accessor.dataframe\_accessor module**

The ADS accessor for the Pandas DataFrame. The accessor will be initialized with the pandas object the user is interacting with.

**Examples**

```
>>> from ads.feature_engineering.accessor.dataframe_accessor import ADSDataFrameAccessor
>>> from ads.feature_engineering.feature_type.continuous import Continuous
>>> from ads.feature_engineering.feature_type.creditcard import CreditCard
>>> from ads.feature_engineering.feature_type.string import String
>>> from ads.feature_engineering.feature_type.base import Tag
>>> df = pd.DataFrame({'Name': ['Alex'], 'CreditCard': ["4532640527811543"]})
>>> df.ads.feature_type
{'Name': ['string'], 'Credit Card': ['string']}
>>> df.ads.feature_type_description
      Column  Feature Type      Description
-----
0      Name      string  Type representing string values.
1  Credit Card      string  Type representing string values.
>>> df.ads.default_type
{'Name': 'string', 'Credit Card': 'string'}
>>> df.ads.feature_type = {'Name': ['string', Tag('abc')]}
>>> df.ads.tags
{'Name': ['abc']}
>>> df.ads.feature_type = {'Credit Card': ['credit_card']}
>>> df.ads.feature_select(include=['credit_card'])
      Credit Card
-----
0      4532640527811543
```

**class** ads.feature\_engineering.accessor.dataframe\_accessor.ADSDataFrameAccessor(*pandas\_obj*)

Bases: [ADSFeatureTypesMixin](#), [EDAMixin](#), [DBAccessMixin](#), [DataLabelingAccessMixin](#)

ADS accessor for the Pandas DataFrame.

**columns**

The column labels of the DataFrame.

**Type**

List[str]

**tags**(self) → Dict[str, str]

Gets the dictionary of user defined tags for the dataframe.

**default\_type**(self) → Dict[str, str]

Gets the map of columns and associated default feature type names.

**feature\_type**(self) → Dict[str, List[str]]

Gets the list of registered feature types.

**feature\_type\_description**(self) → pd.DataFrame

Gets the list of registered feature types in a DataFrame format.

**sync**(self, src: Union[pd.DataFrame, pd.Series]) → pd.DataFrame

Syncs feature types of current DataFrame with that from src.

**feature\_select**(self, include: List[Union[FeatureType, str]] = None, exclude: List[Union[FeatureType, str]] = None) → pd.DataFrame

Gets the list of registered feature types in a DataFrame format.

**help**(self, prop: str = None) → None

Provides docstring for affordable methods and properties.

**Examples**

```
>>> from ads.feature_engineering.accessor.dataframe_accessor import _
↳ ADSDataFrameAccessor
>>> from ads.feature_engineering.feature_type.continuous import Continuous
>>> from ads.feature_engineering.feature_type.creditcard import CreditCard
>>> from ads.feature_engineering.feature_type.string import String
>>> from ads.feature_engineering.feature_type.base import Tag
df = pd.DataFrame({'Name': ['Alex'], 'CreditCard': ["4532640527811543"]})
>>> df.ads.feature_type
{'Name': ['string'], 'Credit Card': ['string']}
>>> df.ads.feature_type_description
      Column  Feature Type  Description
-----
0      Name      string    Type representing string values.
1  Credit Card      string    Type representing string values.
>>> df.ads.default_type
{'Name': 'string', 'Credit Card': 'string'}
>>> df.ads.feature_type = {'Name': ['string', Tag('abc')]}
>>> df.ads.tags
{'Name': ['abc']}
>>> df.ads.feature_type = {'Credit Card': ['credit_card']}
>>> df.ads.feature_select(include=['credit_card'])
      Credit Card
-----
0      4532640527811543
```

Initializes ADS Pandas DataFrame Accessor.

**Parameters**

**pandas\_obj** (*pandas.DataFrame*) – Pandas dataframe

**Raises**

**ValueError** – If provided DataFrame has duplicate columns.

**property default\_type: Dict[str, str]**

Gets the map of columns and associated default feature type names.

**Returns**

The dictionary where key is column name and value is the name of default feature type.

**Return type**

Dict[str, str]

**feature\_select** (*include: Optional[List[Union[FeatureType, str]]] = None, exclude: Optional[List[Union[FeatureType, str]]] = None*) → DataFrame

Returns a subset of the DataFrame's columns based on the column feature\_types.

**Parameters**

- **include** (*List[Union[FeatureType, str]]*, *optional*) – Defaults to None. A list of FeatureType subclass or str to be included.
- **exclude** (*List[Union[FeatureType, str]]*, *optional*) – Defaults to None. A list of FeatureType subclass or str to be excluded.

**Raises**

- **ValueError** – If both of include and exclude are empty
- **ValueError** – If include and exclude are used simultaneously

**Returns**

The subset of the frame including the feature types in include and excluding the feature types in exclude.

**Return type**

pandas.DataFrame

**property feature\_type: Dict[str, List[str]]**

Gets the list of registered feature types.

**Returns**

The dictionary where key is column name and value is list of associated feature type names.

**Return type**

Dict[str, List[str]]

**property feature\_type\_description: DataFrame**

Gets the list of registered feature types in a DataFrame format.

**Return type**

pandas.DataFrame

## Examples

```
>>> df.ads.feature_type_description()
      Column  Feature Type  Description
-----
0      City      string    Type representing string values.
1  Phone Number      string    Type representing string values.
```

**info()** → Any

Gets information about the dataframe.

### Returns

The information about the dataframe.

### Return type

Any

**model\_schema**(*max\_col\_num: int = 2000*)

Generates schema from the dataframe.

### Parameters

**max\_col\_num** (*int, optional. Defaults to 1000*) – The maximum column size of the data that allows to auto generate schema.

## Examples

```
>>> df = pd.read_csv('./orcl_attrition.csv', usecols=['Age', 'Attrition'])
>>> schema = df.ads.model_schema()
>>> schema
Schema:
- description: Attrition
  domain:
    constraints: []
    stats:
      count: 1470
      unique: 2
      values: String
  dtype: object
  feature_type: String
  name: Attrition
  required: true
- description: Age
  domain:
    constraints: []
    stats:
      25%: 31.0
      50%: 37.0
      75%: 44.0
      count: 1470.0
      max: 61.0
      mean: 37.923809523809524
      min: 19.0
      std: 9.135373489136732
```

(continues on next page)

(continued from previous page)

```

        values: Integer
dtype: int64
feature_type: Integer
name: Age
required: true
>>> schema.to_dict()
{'Schema': [{'dtype': 'object',
  'feature_type': 'String',
  'name': 'Attrition',
  'domain': {'values': 'String',
    'stats': {'count': 1470, 'unique': 2},
    'constraints': []},
  'required': True,
  'description': 'Attrition'}],
  {'dtype': 'int64',
  'feature_type': 'Integer',
  'name': 'Age',
  'domain': {'values': 'Integer',
    'stats': {'count': 1470.0,
      'mean': 37.923809523809524,
      'std': 9.135373489136732,
      'min': 19.0,
      '25%': 31.0,
      '50%': 37.0,
      '75%': 44.0,
      'max': 61.0},
    'constraints': []},
  'required': True,
  'description': 'Age'}}}]

```

**Returns**

data schema.

**Return type**

ads.feature\_engineering.schema.Schema

**Raises**

**ads.feature\_engineering.schema.DataSizeTooWide** – If the number of columns of input data exceeds *max\_col\_num*.

**sync**(src: Union[DataFrame, Series]) → DataFrame

Syncs feature types of current DataFrame with that from src.

Syncs feature types of current dataframe with that from src, where src can be a dataframe or a series. In either case, only columns with matched names are synced.

**Parameters**

**src** (pd.DataFrame | pd.Series) – The source to sync from.

**Returns**

Synced dataframe.

**Return type**

pandas.DataFrame

**property tags:** Dict[str, List[str]]

Gets the dictionary of user defined tags for the dataframe. Key is column name and value is list of tag names.

**Returns**

The map of columns and associated default tags.

**Return type**

Dict[str, List[str]]

### 18.1.1.11.5 ads.feature\_engineering.accessor.series\_accessor module

The ADS accessor for the Pandas Series. The accessor will be initialized with the pandas object the user is interacting with.

#### Examples

```
>>> from ads.feature_engineering.accessor.series_accessor import ADSSeriesAccessor
>>> from ads.feature_engineering.feature_type.string import String
>>> from ads.feature_engineering.feature_type.ordinal import Ordinal
>>> from ads.feature_engineering.feature_type.base import Tag
>>> series = pd.Series(['name1', 'name2', 'name3'])
>>> series.ads.default_type
'string'
>>> series.ads.feature_type
['string']
>>> series.ads.feature_type_description
Feature Type          Description
-----
0      string  Type representing string values.
>>> series.ads.feature_type = ['string', Ordinal, Tag('abc')]
>>> series.ads.feature_type
['string', 'ordinal', 'abc']
>>> series1 = series.dropna()
>>> series1.ads.sync(series)
>>> series1.ads.feature_type
['string', 'ordinal', 'abc']
```

**class** ads.feature\_engineering.accessor.series\_accessor.**ADSSeriesAccessor**(pandas\_obj: Series)

Bases: [ADSFeatureTypesMixin](#), [EDAMixinSeries](#)

ADS accessor for Pandas Series.

**name**

The name of Series.

**Type**

str

**tags**

The list of tags for the Series.

**Type**

List[str]

**help**(*self*, *prop*: *str* = *None*) → *None*  
 Provides docstring for affordable methods and properties.

**sync**(*self*, *src*: *Union*[*pd.DataFrame*, *pd.Series*]) → *None*  
 Syncs feature types of current series with that from *src*.

**default\_type**(*self*) → *str*  
 Gets the name of default feature type for the series.

**feature\_type**(*self*) → *List*[*str*]  
 Gets the list of registered feature types for the series.

**feature\_type\_description**(*self*) → *pd.DataFrame*  
 Gets the list of registered feature types in a *DataFrame* format.

## Examples

```
>>> from ads.feature_engineering.accessor.series_accessor import ADSSeriesAccessor
>>> from ads.feature_engineering.feature_type.string import String
>>> from ads.feature_engineering.feature_type.ordinal import Ordinal
>>> from ads.feature_engineering.feature_type.base import Tag
>>> series = pd.Series(['name1', 'name2', 'name3'])
>>> series.ads.default_type
'string'
>>> series.ads.feature_type
['string']
>>> series.ads.feature_type_description
  Feature Type      Description
-----
0      string  Type representing string values.
>>> series.ads.feature_type = ['string', Ordinal, Tag('abc')]
>>> series.ads.feature_type
['string', 'ordinal', 'abc']
>>> series1 = series.dropna()
>>> series1.ads.sync(series)
>>> series1.ads.feature_type
['string', 'ordinal', 'abc']
```

Initializes ADS Pandas Series Accessor.

### Parameters

**pandas\_obj** (*pd.Series*) – The pandas series

### property default\_type: str

Gets the name of default feature type for the series.

### Returns

The name of default feature type.

### Return type

str

### property feature\_type: List[str]

Gets the list of registered feature types for the series.

### Returns

Names of feature types.



**Return type**

List[str]

**Examples**

```
>>> series = pd.Series(['name1'])
>>> series.ads.feature_type = ['name', 'string', Tag('tag for name')]
>>> series.ads.feature_type
['name', 'string', 'tag for name']
```

**property feature\_type\_description: DataFrame**

Gets the list of registered feature types in a DataFrame format.

**Returns**

The DataFrame with feature types for this series.

**Return type**

pd.DataFrame

**Examples**

```
>>> series = pd.Series(['name1'])
>>> series.ads.feature_type = ['name', 'string', Tag('Name tag')]
>>> series.ads.feature_type_description
```

	Feature Type	Description
0	name	Type representing name values.
1	string	Type representing string values.
2	Name tag	Tag.

**sync**(src: Union[DataFrame, Series]) → None

Syncs feature types of current series with that from src.

The src could be a dataframe or a series. In either case, only columns with matched names are synced.

**Parameters**

**src** ((pd.DataFrame | pd.Series)) – The source to sync from.

**Returns**

Nothing.

**Return type**

None

**Examples**

```
>>> series = pd.Series(['name1', 'name2', 'name3', None])
>>> series.ads.feature_type = ['name']
>>> series.ads.feature_type
['name', string]
>>> series.dropna().ads.feature_type
['string']
>>> series1 = series.dropna()
```

(continues on next page)

(continued from previous page)

```
>>> series1.ads.sync(series)
>>> series1.ads.feature_type
['name', 'string']
```

```
class ads.feature_engineering.accessor.series_accessor.ADSSeriesValidator(feature_type_list:
                                                                    List[FeatureType],
                                                                    series: Series)
```

Bases: object

Class helper to invoke registered validator on a series level.

Initializes ADS series validator.

**Parameters**

- **feature\_type\_list** (*List[FeatureType]*) – The list of feature types.
- **series** (*pd.Series*) – The pandas series.

#### 18.1.1.11.6 ads.feature\_engineering.accessor.mixin.correlation module

```
ads.feature_engineering.accessor.mixin.correlation.cat_vs_cat(df: DataFrame, normal_form: bool
                                                            = True) → DataFrame
```

Calculates the correlation of all pairs of categorical features and categorical features.

```
ads.feature_engineering.accessor.mixin.correlation.cat_vs_cont(df: DataFrame,
                                                             categorical_columns,
                                                             continuous_columns,
                                                             normal_form: bool = True) →
DataFrame
```

Calculates the correlation of all pairs of categorical features and continuous features.

```
ads.feature_engineering.accessor.mixin.correlation.cont_vs_cont(df: DataFrame, normal_form:
                                                                bool = True) → DataFrame
```

Calculates the Pearson correlation between two columns of the DataFrame.

#### 18.1.1.11.7 ads.feature\_engineering.accessor.mixin.eda\_mixin module

This exploratory data analysis (EDA) Mixin is used in the ADS accessor for the Pandas Dataframe. The series of purpose-driven methods enable the data scientist to complete analysis on the dataframe.

From the accessor we have access to the pandas object the user is interacting with as well as corresponding lists of feature types per column.

```
class ads.feature_engineering.accessor.mixin.eda_mixin.EDAMixin
```

Bases: object

```
correlation_ratio() → DataFrame
```

Generate a Correlation Ratio data frame for all categorical-continuous variable pairs.

**Returns**

- *pandas.DataFrame*
- *Correlation Ratio correlation data frame with the following 3 columns –*

1. Column 1 (name of the first categorical/continuous column)
2. Column 2 (name of the second categorical/continuous column)
3. Value (correlation value)

---

**Note:** Pairs will be replicated. For example for variables x and y, we would have (x,y), (y,x) both with same correlation value. We will also have (x,x) and (y,y) with value 1.0.

---

**correlation\_ratio\_plot()** → Axes

Generate a heatmap of the Correlation Ratio correlation for all categorical-continuous variable pairs.

**Returns**

Correlation Ratio correlation plot object that can be updated by the customer

**Return type**

Plot object

**cramersv()** → DataFrame

Generate a Cramer's V correlation data frame for all categorical variable pairs.

Gives a warning for dropped non-categorical columns.

**Returns**

**Cramer's V correlation data frame with the following 3 columns:**

1. Column 1 (name of the first categorical column)
2. Column 2 (name of the second categorical column)
3. Value (correlation value)

**Return type**

pandas.DataFrame

---

**Note:** Pairs will be replicated. For example for variables x and y, we would have (x,y), (y,x) both with same correlation value. We will also have (x,x) and (y,y) with value 1.0.

---

**cramersv\_plot()** → Axes

Generate a heatmap of the Cramer's V correlation for all categorical variable pairs.

Gives a warning for dropped non-categorical columns.

**Returns**

Cramer's V correlation plot object that can be updated by the customer

**Return type**

Plot object

**feature\_count()** → DataFrame

Counts the number of columns for each feature type and each primary feature. The column of primary is the number of primary feature types that is assigned to the column.

**Returns**

The number of columns for each feature type The number of columns for each primary feature

**Return type**

Dataframe with

## Examples

```
>>> df.ads.feature_type
{'PassengerId': ['ordinal', 'category'],
 'Survived': ['ordinal'],
 'Pclass': ['ordinal'],
 'Name': ['category'],
 'Sex': ['category']}
>>> df.ads.feature_count()
  Feature Type  Count  Primary
0    category      3         2
1    ordinal      3         3
```

**feature\_plot()** → DataFrame

For every column in the dataframe plot generate a list of summary plots based on the most relevant feature type.

### Returns

Dataframe with 2 columns: 1. Column - feature name 2. Plot - plot object

### Return type

pandas.DataFrame

**feature\_stat()** → DataFrame

Summary statistics Dataframe provided.

This returns feature stats on each column using FeatureType summary method.

## Examples

```
>>> df = pd.read_csv('~/.advanced-ds/tests/vor_datasets/vor_titanic.csv')
>>> df.ads.feature_stat().head()
  Column  Metric  Value
0  PassengerId  count    891.000
1  PassengerId  mean    446.000
2  PassengerId  standard deviation  257.354
3  PassengerId  sample minimum    1.000
4  PassengerId  lower quartile    223.500
```

### Returns

Dataframe with 3 columns: name, metric, value

### Return type

pandas.DataFrame

**pearson()** → DataFrame

Generate a Pearson correlation data frame for all continuous variable pairs.

Gives a warning for dropped non-numerical columns.

### Returns

- pandas.DataFrame
- *Pearson correlation data frame with the following 3 columns –*

1. Column 1 (name of the first continuous column)
2. Column 2 (name of the second continuous column)
3. Value (correlation value)

---

**Note:** Pairs will be replicated. For example for variables x and y, we'd have (x,y), (y,x) both with same correlation value. We'll also have (x,x) and (y,y) with value 1.0.

---

**pearson\_plot()** → Axes

Generate a heatmap of the Pearson correlation for all continuous variable pairs.

**Returns**

Pearson correlation plot object that can be updated by the customer

**Return type**

Plot object

**warning()** → DataFrame

Generates a data frame that lists feature specific warnings.

**Returns**

The list of feature specific warnings.

**Return type**

pandas.DataFrame

## Examples

```
>>> df.ads.warning()
  Column  Feature Type  Warning  Message  Metric
↪ Value
-----
↪ -----
0      Age    continuous    Zeros    Age has 38 zeros    Count
↪      38
1      Age    continuous    Zeros    Age has 12.2% zeros  Percentage
↪    12.2%
```

### 18.1.1.11.8 ads.feature\_engineering.accessor.mixin.eda\_mixin\_series module

This exploratory data analysis (EDA) Mixin is used in the ADS accessor for the Pandas Series. The series of purpose-driven methods enable the data scientist to complete univariate analysis.

From the accessor we have access to the pandas object the user is interacting with as well as corresponding list of feature types.

**class** ads.feature\_engineering.accessor.mixin.eda\_mixin\_series.EDAMixinSeries

Bases: object

**feature\_plot()** → Axes

For the series generate a summary plot based on the most relevant feature type.

**Returns**

Plot object for the series based on the most relevant feature type.

**Return type**

matplotlib.axes.\_subplots.AxesSubplot

**feature\_stat()** → DataFrame

Summary statistics Dataframe provided.

This returns feature stats on series using FeatureType summary method.

**Examples**

```
>>> df = pd.read_csv('~/.advanced-ds/tests/vor_datasets/vor_titanic.csv')
>>> df['Cabin'].ads.feature_stat()
   Metric  Value
0   count    891
1  unique    147
2  missing    687
```

**Returns**

Dataframe with 2 columns and rows for different metric values

**Return type**

pandas.DataFrame

**warning()** → DataFrame

Generates a data frame that lists feature specific warnings.

**Returns**

The list of feature specific warnings.

**Return type**

pandas.DataFrame

**Examples**

```
>>> df["Age"].ads.warning()
   Feature Type  Warning      Message  Metric  Value
-----
0  continuous    Zeros  Age has 38 zeros    Count    38
1  continuous    Zeros  Age has 12.2% zeros  Percentage  12.2%
```

**18.1.1.11.9 ads.feature\_engineering.accessor.mixin.feature\_types\_mixin module**

The module that represents the ADS Feature Types Mixin class that extends Pandas Series and Dataframe accessors.

Classes

**ADSFeatureTypesMixin**

ADS Feature Types Mixin class that extends Pandas Series and Dataframe accessors.

**class** `ads.feature_engineering.accessor.mixin.feature_types_mixin.ADSFeatureTypesMixin`

Bases: object

ADS Feature Types Mixin class that extends Pandas Series and DataFrame accessors.

**warning\_registered**(*cls*) → `pd.DataFrame`

Lists registered warnings for registered feature types.

**validator\_registered**(*cls*) → `pd.DataFrame`

Lists registered validators for registered feature types.

**help**(*self*, *prop*: *str* = *None*) → *None*

Help method that prints either a table of available properties or, given a property, returns its docstring.

**help**(*prop*: *Optional*[*str*] = *None*) → *None*

Help method that prints either a table of available properties or, given an individual property, returns its docstring.

**Parameters**

**prop** (*str*) – The Name of property.

**Returns**

Nothing.

**Return type**

*None*

**validator\_registered**() → `DataFrame`

Lists registered validators for registered feature types.

**Returns**

The list of registered validators for registered feature types

**Return type**

`pandas.DataFrame`

Examples

>>> df.ads.validator_registered()					
	Column	Feature Type	Validator	Condition	
	Handler				
	-----				
0	PhoneNumber	phone_number	is_phone_number	()	
	default_handler				
1	PhoneNumber	phone_number	is_phone_number	{'country_code': '+7'}	
	specific_country_handler				
2	CreditCard	credit_card	is_credit_card	()	
	default_handler				

```
>>> df['PhoneNumber'].ads.validator_registered()
Feature Type      Validator      Condition
Handler
-----
0  phone_number    is_phone_number      ()
default_handler
1  phone_number    is_phone_number  {'country_code': '+7'}  specific_
country_handler
```

**warning\_registered()** → DataFrame

Lists registered warnings for all registered feature types.

**Returns**

The list of registered warnings for registered feature types.

**Return type**

pandas.DataFrame

**Examples**

```
>>> df.ads.warning_registered()
Column  Feature Type      Warning      Handler
-----
0  Age      continuous      zeros      zeros_handler
1  Age      continuous  high_cardinality  high_cardinality_handler

>>> df["Age"].ads.warning_registered()
Feature Type      Warning      Handler
-----
0  continuous      zeros      zeros_handler
1  continuous  high_cardinality  high_cardinality_handler
```

#### 18.1.1.11.10 ads.feature\_engineering.adsstring.common\_regex\_mixin module

**class** ads.feature\_engineering.adsstring.common\_regex\_mixin.CommonRegexMixin

Bases: object

**property** address

**property** credit\_card

**property** date

**property** email

**property** ip

**property** link

**property** phone\_number\_US

**property** price



**redact**(*fields*: Union[List[str], Dict[str, str]]) → str

Remove personal information in a string. For example, “Jane’s phone number is 123-456-7890” is turned into “Jane’s phone number is [phone\_number\_US].”

**Parameters**

**fields** ((*list(str)* | *dict*)) – either a list of fields to redact, e.g. ['email', 'phone\_number\_US'], in which case the redacted text is replaced with capitalized word like [EMAIL] or [PHONE\_NUMBER\_US\_WITH\_EXT], or a dictionary where key is a field to redact and value is the replacement text, e.g., {'email': 'HIDDEN\_EMAIL'}.

**Returns**

redacted string

**Return type**

str

```
redact_map = {'address': '[ADDRESS]', 'address_with_zip': '[ADDRESS_WITH_ZIP]',
'credit_card': '[CREDIT_CARD]', 'date': '[DATE]', 'email': '[EMAIL]', 'ip': '[IP]',
'ipv6': '[IPV6]', 'link': '[LINK]', 'phone_number_US': '[PHONE_NUMBER_US]',
'phone_number_US_with_ext': '[PHONE_NUMBER_US_WITH_EXT]', 'po_box': '[PO_BOX]',
'price': '[PRICE]', 'ssn': '[SSN]', 'time': '[TIME]', 'zip_code': '[ZIP_CODE]'}
```

property ssn

property time

property zip\_code

#### 18.1.1.11.11 ads.feature\_engineering.adsstring.oci\_language module

#### 18.1.1.11.12 ads.feature\_engineering.adsstring.string module

#### 18.1.1.11.13 ads.feature\_engineering.feature\_type.address module

The module that represents an Address feature type.

**Classes:**

**Address**

The Address feature type.

**class** ads.feature\_engineering.feature\_type.address.**Address**

Bases: *String*

Type representing address.

**description**

The feature type description.

**Type**

str

**name**

The feature type name.

**Type**

str

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x: pd.Series*) → *pd.DataFrame*

Generates feature statistics.

**feature\_plot**(*x: pd.Series*) → *plt.Axes*

Shows the location of given address on map base on zip code.

**Example**

```
>>> from ads.feature_engineering.feature_type.address import Address
>>> import pandas as pd
>>> address = pd.Series(['1 Miller Drive, New York, NY 12345',
                        '1 Berkeley Street, Boston, MA 67891',
                        '54305 Oxford Street, Seattle, WA 95132',
                        ''])
>>> Address.validator.is_address(address)
0      True
1      True
2      True
3     False
dtype: bool
```

**description** = 'Type representing address.'

**classmethod feature\_domain**(*x: Series*) → *Domain*

Generate the domain of the data of this feature type.

**Examples**

```
>>> address = pd.Series(['1 Miller Drive, New York, NY 12345',
                        '1 Berkeley Street, Boston, MA 67891',
                        '54305 Oxford Street, Seattle, WA 95132',
                        ''],
                        name='address')
>>> address.ads.feature_type = ['address']
>>> address.ads.feature_domain()
constraints: []
stats:
  count: 4
  missing: 1
  unique: 3
values: Address
```

**Returns**

Domain based on the Address feature type.

**Return type**

ads.feature\_engineering.schema.Domain

**static feature\_plot**(*x: Series*) → Axes

Shows the location of given address on map base on zip code.

**Examples**

```
>>> address = pd.Series(['1 Miller Drive, New York, NY 12345',
                        '1 Berkeley Street, Boston, MA 67891',
                        '54305 Oxford Street, Seattle, WA 95132',
                        ''],
                        name='address')
>>> address.ads.feature_type = ['address']
>>> address.ads.feature_plot()
```

**Returns**

Plot object for the series based on the Address feature type.

**Return type**

matplotlib.axes.\_subplots.AxesSubplot

**static feature\_stat**(*x: Series*) → DataFrame

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

**Examples**

```
>>> address = pd.Series(['1 Miller Drive, New York, NY 12345',
                        '1 Berkeley Street, Boston, MA 67891',
                        '54305 Oxford Street, Seattle, WA 95132',
                        ''],
                        name='address')
>>> address.ads.feature_type = ['address']
>>> address.ads.feature_stat()
Metric Value
0      count    4
1     unique    3
2     missing    1
```

**Returns**

Summary statistics of the Series provided.

**Return type**

pandas.DataFrame

**validator =**

```
<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator
object>
```

```
warning =  
<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>
```

```
ads.feature_engineering.feature_type.address.default_handler(data: Series, *args, **kwargs) →  
Series
```

Processes given data and indicates if the data matches requirements.

**Parameters**

**data** (*pd.Series*) – The data to process.

**Returns**

The logical list indicating if the data matches requirements.

**Return type**

`pandas.Series`

#### 18.1.1.11.14 `ads.feature_engineering.feature_type.base` module

```
class ads.feature_engineering.feature_type.base.FeatureBaseType(classname, bases, dictionary)  
Bases: type
```

The helper metaclass to extend functionality of FeatureType class.

```
class ads.feature_engineering.feature_type.base.FeatureBaseTypeMeta(classname, bases,  
dictionary)
```

Bases: `FeatureBaseType`, `ABCMeta`

The class to provide compatibility between ABC and FeatureBaseType metaclass.

```
class ads.feature_engineering.feature_type.base.FeatureType  
Bases: ABC
```

Abstract case for feature types. Default class attribute include name and description. Name is auto generated using camel to snake conversion unless specified.

```
description = 'Base feature type.'
```

```
name = 'feature_type'
```

```
validator =
```

```
<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator  
object>
```

```
warning =
```

```
<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>
```

```
class ads.feature_engineering.feature_type.base.Name  
Bases: object
```

```
class ads.feature_engineering.feature_type.base.Tag(name: str)  
Bases: object
```

Class for free form tags. Name must be specified.

Initialize a tag instance.

**Parameters**

**name** (*str*) – The name of the tag.

### 18.1.1.11.15 ads.feature\_engineering.feature\_type.boolean module

The module that represents a Boolean feature type.

#### Classes:

##### **Boolean**

The feature type that represents binary values True/False.

#### Functions:

##### **default\_handler(data: pd.Series) -> pd.Series**

Processes given data and indicates if the data matches requirements.

##### **class ads.feature\_engineering.feature\_type.boolean.Boolean**

Bases: *FeatureType*

Type representing binary values True/False.

##### **description**

The feature type description.

##### **Type**

str

##### **name**

The feature type name.

##### **Type**

str

##### **warning**

Provides functionality to register warnings and invoke them.

##### **Type**

*FeatureWarning*

##### **validator**

Provides functionality to register validators and invoke them.

##### **feature\_stat(x: pd.Series) → pd.DataFrame**

Generates feature statistics.

##### **feature\_plot(x: pd.Series) → plt.Axes**

Show the counts of observations in True/False using bars.

#### Examples

```
>>> from ads.feature_engineering.feature_type.boolean import Boolean
>>> import pandas as pd
>>> import numpy as np
>>> s = pd.Series([True, False, True, False, np.NaN, None], name='bool')
>>> s.ads.feature_type = ['boolean']
>>> Boolean.validator.is_boolean(s)
0      True
1      True
2      True
3      True
```

(continues on next page)

(continued from previous page)

```

4    False
5    False
dtype: bool

```

**description** = 'Type representing binary values True/False.'

**classmethod** `feature_domain(x: Series) → Domain`

Generate the domain of the data of this feature type.

## Examples

```

>>> s = pd.Series([True, False, True, False, np.NaN, None], name='bool')
>>> s.ads.feature_type = ['boolean']
>>> s.ads.feature_domain()
constraints:
- expression: $x in [True, False]
  language: python
stats:
  count: 6
  missing: 2
  unique: 2
values: Boolean

```

### Returns

Domain based on the Boolean feature type.

### Return type

`ads.feature_engineering.schema.Domain`

**static** `feature_plot(x: Series) → Axes`

Shows the counts of observations in True/False using bars.

### Parameters

**x** (`pandas.Series`) – The feature being evaluated.

### Returns

Plot object for the series based on the Boolean feature type.

### Return type

`matplotlib.axes._subplots.AxesSubplot`

## Examples

```

>>> s = pd.Series([True, False, True, False, np.NaN, None], name='bool')
>>> s.ads.feature_type = ['boolean']
>>> s.ads.feature_plot()

```

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

**Parameters**

**x** (pandas.Series) – The feature being evaluated.

**Returns**

Summary statistics of the Series or Dataframe provided.

**Return type**

pandas.DataFrame

**Examples**

```
>>> s = pd.Series([True, False, True, False, np.NaN, None], name='bool')
>>> s.ads.feature_type = ['boolean']
>>> s.ads.feature_stat()
Metric Value
0      count  6
1     unique  2
2    missing  2
```

**validator =**

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning =**

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

ads.feature\_engineering.feature\_type.boolean.default\_handler(*data: Series, \*args, \*\*kwargs*) → Series

Processes given data and indicates if the data matches requirements.

**Parameters**

**data** (pandas.Series) – The data to process.

**Returns**

The logical list indicating if the data matches requirements.

**Return type**

pandas.Series

**18.1.1.11.16 ads.feature\_engineering.feature\_type.category module**

The module that represents a Category feature type.

**Classes:****Category**

The Category feature type.

**class** ads.feature\_engineering.feature\_type.category.Category

Bases: *FeatureType*

Type representing discrete unordered values.

**description**

The feature type description.

**Type**

str

**name**

The feature type name.

**Type**

str

**warning**

Provides functionality to register warnings and invoke them.

**Type***FeatureWarning***validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x*: *pd.Series*) → *pd.DataFrame*

Generates feature statistics.

**feature\_plot**(*x*: *pd.Series*) → *plt.Axes*

Shows the counts of observations in each categorical bin using bar chart.

**description** = 'Type representing discrete unordered values.'**classmethod feature\_domain**(*x*: *Series*) → *Domain*

Generate the domain of the data of this feature type.

**Examples**

```
>>> cat = pd.Series(['S', 'C', 'S', 'S', 'S', 'Q', 'S', 'S', 'S', 'C', 'S', 'S',  
↳ 'S',  
                    'S', 'S', 'S', 'Q', 'S', 'S', '', np.NaN, None], name='category')  
>>> cat.ads.feature_type = ['category']  
>>> cat.ads.feature_domain()  
constraints:  
- expression: $x in ['S', 'C', 'Q', '']  
  language: python  
stats:  
  count: 22  
  missing: 3  
  unique: 3  
values: Category
```

**Returns**

Domain based on the Category feature type.

**Return type***ads.feature\_engineering.schema.Domain***static feature\_plot**(*x*: *Series*) → *Axes*

Shows the counts of observations in each categorical bin using bar chart.

**Parameters****x** (*pandas.Series*) – The feature being evaluated.



**Returns**

Plot object for the series based on the Category feature type.

**Return type**

matplotlib.axes.\_subplots.AxesSubplot

**Examples**

```
>>> cat = pd.Series(['S', 'C', 'S', 'S', 'S', 'Q', 'S', 'S', 'S', 'C', 'S', 'S',
↪ 'S',
                    'S', 'S', 'S', 'Q', 'S', 'S', '', np.NaN, None], name='category')
>>> cat.ads.feature_type = ['category']
>>> cat.ads.feature_plot()
```

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count) if there are any.

**Parameters**

**x** (pandas.Series) – The feature being evaluated.

**Returns**

Summary statistics of the Series or Dataframe provided.

**Return type**

pandas.DataFrame

**Examples**

```
>>> cat = pd.Series(['S', 'C', 'S', 'S', 'S', 'Q', 'S', 'S', 'S', 'C', 'S', 'S',
↪ 'S',
                    'S', 'S', 'S', 'Q', 'S', 'S', '', np.NaN, None], name='category')
>>> cat.ads.feature_type = ['category']
>>> cat.ads.feature_stat()
Metric Value
0      count  22
1     unique   3
2     missing   3
```

**validator =**

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning =**

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

#### 18.1.1.11.17 ads.feature\_engineering.feature\_type.constant module

The module that represents a Constant feature type.

##### Classes:

###### **Constant**

The Constant feature type.

**class** ads.feature\_engineering.feature\_type.constant.**Constant**

Bases: *FeatureType*

Type representing constant values.

###### **description**

The feature type description.

###### **Type**

str

###### **name**

The feature type name.

###### **Type**

str

###### **warning**

Provides functionality to register warnings and invoke them.

###### **Type**

*FeatureWarning*

###### **validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x: pd.Series*) → pd.DataFrame

Generates feature statistics.

**feature\_plot**(*x: pd.Series*) → plt.Axes

Shows the counts of observations in bars.

**description** = 'Type representing constant values.'

**classmethod feature\_domain**(*x: Series*) → Domain

Generate the domain of the data of this feature type. .. rubric:: Example

```
>>> s = pd.Series([1, 1, 1, 1, 1], name='constant')
>>> s.ads.feature_type = ['constant']
>>> s.ads.feature_domain()
constraints: []
stats:
  count: 5
  unique: 1
values: Constant
```

###### **Returns**

Domain based on the Constant feature type.

**Return type**

ads.feature\_engineering.schema.Domain

**static feature\_plot**(*x: Series*) → Axes

Shows the counts of observations in bars.

**Parameters****x** (pandas.Series) – The feature being shown.**Examples**

```
>>> s = pd.Series([1, 1, 1, 1, 1], name='constant')
>>> s.ads.feature_type = ['constant']
>>> s.ads.feature_plot()
```

**Returns**

Plot object for the series based on the Constant feature type.

**Return type**

matplotlib.axes.\_subplots.AxesSubplot

**static feature\_stat**(*x: Series*) → DataFrame

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

**Parameters****x** (pandas.Series) – The feature being evaluated.**Returns**

Summary statistics of the Series provided.

**Return type**

pandas.DataFrame

**Examples**

```
>>> s = pd.Series([1, 1, 1, 1, 1], name='constant')
>>> s.ads.feature_type = ['constant']
>>> s.ads.feature_stat()
Metric Value
0      count    5
1      unique    1
```

validator =

```
<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator
object>
```

warning =

```
<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>
```

### 18.1.1.11.18 ads.feature\_engineering.feature\_type.continuous module

The module that represents a Continuous feature type.

#### Classes:

##### **Continuous**

The Continuous feature type.

**class** ads.feature\_engineering.feature\_type.continuous.**Continuous**

Bases: *FeatureType*

Type representing continuous values.

#### **description**

The feature type description.

##### **Type**

str

#### **name**

The feature type name.

##### **Type**

str

#### **warning**

Provides functionality to register warnings and invoke them.

##### **Type**

*FeatureWarning*

#### **validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x: pd.Series*) → pd.DataFrame

Generates feature statistics.

**feature\_plot**(*x: pd.Series*) → plt.Axes

Shows distributions of datasets using box plot.

**description** = 'Type representing continuous values.'

**classmethod feature\_domain**(*x: Series*) → Domain

Generate the domain of the data of this feature type.

#### **Examples**

```
>>> cts = pd.Series([13.32, 3.32, 4.3, 2.45, 6.34, 2.25,
                    4.43, 3.26, np.NaN, None], name='continuous')
>>> cts.ads.feature_type = ['continuous']
>>> cts.ads.feature_domain()
constraints: []
stats:
  count: 10.0
  lower quartile: 3.058
  mean: 4.959
```

(continues on next page)

(continued from previous page)

```

median: 3.81
missing: 2.0
sample maximum: 13.32
sample minimum: 2.25
skew: 2.175
standard deviation: 3.62
upper quartile: 4.908
values: Continuous

```

**Returns**

Domain based on the Continuous feature type.

**Return type**

`ads.feature_engineering.schema.Domain`

**static** `feature_plot(x: Series) → Axes`

Shows distributions of datasets using box plot.

**Examples**

```

>>> cts = pd.Series([13.32, 3.32, 4.3, 2.45, 6.34, 2.25,
                    4.43, 3.26, np.NaN, None], name='continuous')
>>> cts.ads.feature_type = ['continuous']
>>> cts.ads.feture_plot()

```

**Returns**

Plot object for the series based on the Continuous feature type.

**Return type**

`matplotlib.axes._subplots.AxesSubplot`

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, mean, standard deviation, sample minimum, lower quartile, median, 75%, upper quartile, skew and missing(count).

**Examples**

```

>>> cts = pd.Series([13.32, 3.32, 4.3, 2.45, 6.34, 2.25,
                    4.43, 3.26, np.NaN, None], name='continuous')
>>> cts.ads.feature_type = ['continuous']
>>> cts.ads.feature_stat()

```

	Metric	Value
0	count	10.000
1	mean	4.959
2	standard deviation	3.620
3	sample minimum	2.250
4	lower quartile	3.058
5	median	3.810

(continues on next page)

(continued from previous page)

6	upper quartile	4.908
7	sample maximum	13.320
8	skew	2.175
9	missing	2.000

**Returns**

Summary statistics of the Series or Dataframe provided.

**Return type**

`pandas.DataFrame`

**validator =**

`<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator object>`

**warning =**

`<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>`

#### 18.1.1.11.19 `ads.feature_engineering.feature_type.creditcard` module

The module that represents a CreditCard feature type.

**Classes:****CreditCard**

The CreditCard feature type.

**Functions:**

**default\_handler(data: `pd.Series`) -> `pd.Series`**

Processes given data and indicates if the data matches requirements.

**\_luhn\_checksum(card\_number: `str`) -> `float`**

Implements Luhn algorithm to validate a credit card number.

**class `ads.feature_engineering.feature_type.creditcard.CreditCard`**

Bases: *`String`*

Type representing credit card numbers.

**description**

The feature type description.

**Type**

`str`

**name**

The feature type name.

**Type**

`str`

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*`FeatureWarning`*

**validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x*: *pd.Series*) → *pd.DataFrame*

Generates feature statistics.

**feature\_plot**(*x*: *pd.Series*) → *plt.Axes*

Shows the counts of observations in each credit card type using bar chart.

**Examples**

```
>>> from ads.feature_engineering.feature_type.creditcard import CreditCard
>>> import pandas as pd
>>> s = pd.Series(["4532640527811543", None, "4556929308150929", "4539944650919740",
→ "4485348152450846", "4556593717607190"], name='credit_card')
>>> s.ads.feature_type = ['credit_card']
>>> CreditCard.validator.is_credit_card(s)
0      True
1     False
2      True
3      True
4      True
5      True
Name: credit_card, dtype: bool
```

**description** = 'Type representing credit card numbers.'

**classmethod feature\_domain**(*x*: *Series*) → *Domain*

Generate the domain of the data of this feature type.

**Examples**

```
>>> visa = [
    "4532640527811543",
    None,
    "4556929308150929",
    "4539944650919740",
    "4485348152450846",
    "4556593717607190",
]
>>> mastercard = [
    "5334180299390324",
    "5111466404826446",
    "5273114895302717",
    "543097215222336",
    "5536426859893306",
]
>>> amex = [
    "371025944923273",
    "374745112042294",
    "340984902710890",
```

(continues on next page)

(continued from previous page)

```

    "375767928645325",
    "370720852891659",
    ]
>>> creditcard_list = visa + mastercard + amex
>>> creditcard_series = pd.Series(creditcard_list,name='card')
>>> creditcard_series.ads.feature_type = ['credit_card']
>>> creditcard_series.ads.feature_domain()
constraints: []
stats:
  count: 16
  count_Amex: 5
  count_Diners Club: 2
  count_MasterCard: 3
  count_Visa: 5
  count_missing: 1
  missing: 1
  unique: 15
values: CreditCard

```

**Returns**

Domain based on the CreditCard feature type.

**Return type**

ads.feature\_engineering.schema.Domain

**static feature\_plot**(*x: Series*) → Axes

Shows the counts of observations in each credit card type using bar chart.

**Examples**

```

>>> visa = [
    "4532640527811543",
    None,
    "4556929308150929",
    "4539944650919740",
    "4485348152450846",
    "4556593717607190",
    ]
>>> mastercard = [
    "5334180299390324",
    "5111466404826446",
    "5273114895302717",
    "543097215222336",
    "5536426859893306",
    ]
>>> amex = [
    "371025944923273",
    "374745112042294",
    "340984902710890",
    "375767928645325",
    "370720852891659",
    ]

```

(continues on next page)



(continued from previous page)

```

    ]
    >>> creditcard_list = visa + mastercard + amex
    >>> creditcard_series = pd.Series(creditcard_list,name='card')
    >>> creditcard_series.ads.feature_type = ['credit_card']
    >>> creditcard_series.ads.feature_plot()

```

**Returns**

Plot object for the series based on the CreditCard feature type.

**Return type**

matplotlib.axes.\_subplots.AxesSubplot

**static feature\_stat(*x: Series*)**

Generates feature statistics.

**Feature statistics include (total)count, unique(count), missing(count) and count of each credit card type.**

**Examples**

```

>>> visa = [
    "4532640527811543",
    None,
    "4556929308150929",
    "4539944650919740",
    "4485348152450846",
    "4556593717607190",
    ]
>>> mastercard = [
    "5334180299390324",
    "5111466404826446",
    "5273114895302717",
    "543097215222336",
    "5536426859893306",
    ]
>>> amex = [
    "371025944923273",
    "374745112042294",
    "340984902710890",
    "375767928645325",
    "370720852891659",
    ]
>>> creditcard_list = visa + mastercard + amex
>>> creditcard_series = pd.Series(creditcard_list,name='card')
>>> creditcard_series.ads.feature_type = ['credit_card']
>>> creditcard_series.ads.feature_stat()

```

	Metric	Value
0	count	16
1	unique	15
2	missing	1
3	count_Amex	5

(continues on next page)

(continued from previous page)

4	count_Visa	5
5	count_MasterCard	3
6	count_Diners Club	2
7	count_missing	1

**Returns**

Summary statistics of the Series or Dataframe provided.

**Return type**

`pandas.DataFrame`

**validator =**

`<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator object>`

**warning =**

`<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>`

`ads.feature_engineering.feature_type.creditcard.default_handler(data: Series, *args, **kwargs)`  
→ Series

Processes given data and indicates if the data matches requirements.

**Parameters**

**data** (`pandas.Series`) – The data to process.

**Returns**

The logical list indicating if the data matches requirements.

**Return type**

`pandas.Series`

#### 18.1.1.11.20 `ads.feature_engineering.feature_type.datetime` module

The module that represents a DateTime feature type.

**Classes:****DateTime**

The DateTime feature type.

**class** `ads.feature_engineering.feature_type.datetime.DateTime`

Bases: `FeatureType`

Type representing date and/or time.

**description**

The feature type description.

**Type**

`str`

**name**

The feature type name.

**Type**

`str`

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x: pd.Series*) → *pd.DataFrame*

Generates feature statistics.

**feature\_plot**(*x: pd.Series*) → *plt.Axes*

Shows distributions of datetime datasets using histograms.

**Example**

```
>>> from ads.feature_engineering.feature_type.datetime import DateTime
>>> import pandas as pd
>>> s = pd.Series(["12/12/12", "12/12/13", None, "12/12/14"], name='datetime')
>>> s.ads.feature_type = ['date_time']
>>> DateTime.validator.is_datetime(s)
0      True
1      True
2     False
3      True
Name: datetime, dtype: bool
```

**description** = 'Type representing date and/or time.'

**classmethod feature\_domain**(*x: Series*) → *Domain*

Generate the domain of the data of this feature type.

**Examples**

```
>>> s = pd.Series(['3/11/2000', '3/12/2000', '3/13/2000', '', None, np.nan,
↳ 'April/13/2011', 'April/15/11'], name='datetime')
>>> s.ads.feature_type = ['date_time']
>>> s.ads.feature_domain()
constraints: []
stats:
  count: 8
  missing: 3
  sample maximum: April/15/11
  sample minimum: 3/11/2000
values: DateTime
```

**Returns**

Domain based on the DateTime feature type.

**Return type**

*ads.feature\_engineering.schema.Domain*

**static** `feature_plot(x: Series) → Axes`

Shows distributions of datetime datasets using histograms.

### Examples

```
>>> x = pd.Series(['3/11/2000', '3/12/2000', '3/13/2000', '', None, np.nan,
↳ 'April/13/2011', 'April/15/11'], name='datetime')
>>> x.ads.feature_type = ['date_time']
>>> x.ads.feature_plot()
```

#### Returns

Plot object for the series based on the DateTime feature type.

#### Return type

`matplotlib.axes._subplots.AxesSubplot`

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, sample maximum, sample minimum, and missing(count) if there is any.

### Examples

```
>>> x = pd.Series(['3/11/2000', '3/12/2000', '3/13/2000', '', None, np.nan,
↳ 'April/13/2011', 'April/15/11'], name='datetime')
>>> x.ads.feature_type = ['date_time']
>>> x.ads.feature_stat()
  Metric      Value
0   count          8
1  sample maximum  April/15/11
2  sample minimum   3/11/2000
3   missing          3
```

#### Returns

Summary statistics of the Series or Dataframe provided.

#### Return type

`pandas.DataFrame`

**validator =**

`<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator object>`

**warning =**

`<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>`

`ads.feature_engineering.feature_type.datetime.default_handler(data: Series, *args, **kwargs) → Series`

Processes given data and indicates if the data matches requirements.

#### Parameters

**data** (`pandas.Series`) – The data to process.

**Returns**

The logical list indicating if the data matches requirements.

**Return type**

`pandas.Series`

**18.1.1.11.21 `ads.feature_engineering.feature_type.discrete` module**

The module that represents a Discrete feature type.

**Classes:****Discrete**

The Discrete feature type.

**class** `ads.feature_engineering.feature_type.discrete.Discrete`

Bases: *FeatureType*

Type representing discrete values.

**description**

The feature type description.

**Type**

`str`

**name**

The feature type name.

**Type**

`str`

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x: pd.Series*) → `pd.DataFrame`

Generates feature statistics.

**feature\_plot**(*x: pd.Series*) → `plt.Axes`

Shows distributions of datasets using box plot.

**description** = 'Type representing discrete values.'

**classmethod** **feature\_domain**(*x: Series*) → `Domain`

Generate the domain of the data of this feature type.

### Examples

```
>>> discrete_numbers = pd.Series([35, 25, 13, 42],
                                name='discrete')
>>> discrete_numbers.ads.feature_type = ['discrete']
>>> discrete_numbers.ads.feature_domain()
constraints: []
stats:
  count: 4
  unique: 4
values: Discrete
```

#### Returns

Domain based on the Discrete feature type.

#### Return type

`ads.feature_engineering.schema.Domain`

**static** `feature_plot(x: Series) → Axes`

Shows distributions of datasets using box plot.

### Examples

```
>>> discrete_numbers = pd.Series([35, 25, 13, 42],
                                name='discrete')
>>> discrete_numbers.ads.feature_type = ['discrete']
>>> discrete_numbers.ads.feature_stat()
Metric Value
0      count    4
1     unique    4
```

#### Returns

Plot object for the series based on the Discrete feature type.

#### Return type

`matplotlib.axes._subplots.AxesSubplot`

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

### Examples

```
>>> discrete_numbers = pd.Series([35, 25, 13, 42],
                                name='discrete')
>>> discrete_numbers.ads.feature_type = ['discrete']
>>> discrete_numbers.ads.feature_stat()
discrete
count    4
unique   4
```

**Returns**

Summary statistics of the Series provided.

**Return type**

`pandas.DataFrame`

`validator =`

`<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator object>`

`warning =`

`<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>`

**18.1.1.11.22 ads.feature\_engineering.feature\_type.document module**

The module that represents a Document feature type.

**Classes:****Document**

The Document feature type.

`class ads.feature_engineering.feature_type.document.Document`

Bases: *FeatureType*

Type representing document values.

**description**

The feature type description.

**Type**

`str`

**name**

The feature type name.

**Type**

`str`

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

`description = 'Type representing document values.'`

`classmethod feature_domain()`

**Returns**

Nothing.

**Return type**

`None`

```
validator =  
<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator  
object>  
  
warning =  
<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>
```

#### 18.1.1.11.23 ads.feature\_engineering.feature\_type.gis module

The module that represents a GIS feature type.

Classes:

##### **GIS**

The GIS feature type.

**class** ads.feature\_engineering.feature\_type.gis.GIS

Bases: *FeatureType*

Type representing geographic information.

##### **description**

The feature type description.

##### **Type**

str

##### **name**

The feature type name.

##### **Type**

str

##### **warning**

Provides functionality to register warnings and invoke them.

##### **Type**

*FeatureWarning*

##### **validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x: pd.Series*) → pd.DataFrame

Generates feature statistics.

**feature\_plot**(*x: pd.Series*) → plt.Axes

Shows the location of given address on map base on longitude and latitude.



### Example

```

>>> from ads.feature_engineering.feature_type.gis import GIS
>>> import pandas as pd
>>> s = pd.Series(["-18.2193965, -93.587285",
                  "-21.0255305, -122.478584",
                  "85.103913, 19.405744",
                  "82.913736, 178.225672",
                  "62.9795085, -66.989705",
                  "54.5604395, 95.235090",
                  "24.2811855, -162.380403",
                  "-1.818319, -80.681214",
                  None,
                  "(51.816119, 175.979008)",
                  "(54.3392995, -11.801615)"],
                  name='gis')
>>> s.ads.feature_type = ['gis']
>>> GIS.validator.is_gis(s)
0      True
1      True
2      True
3      True
4      True
5      True
6      True
7      True
8     False
9      True
10     True
Name: gis, dtype: bool

```

**description** = 'Type representing geographic information.'

**classmethod** `feature_domain(x: Series) → Domain`

Generate the domain of the data of this feature type.

### Examples

```

>>> gis = pd.Series([
    "69.196241, -125.017615",
    "5.2272595, -143.465712",
    "-33.9855425, -153.445155",
    "43.340610, 86.460554",
    "24.2811855, -162.380403",
    "2.7849025, -7.328156",
    "45.033805, 157.490179",
    "-1.818319, -80.681214",
    "-44.510428, -169.269477",
    "-56.3344375, -166.407038",
    "",
    np.NaN,
    None
])

```

(continues on next page)

(continued from previous page)

```

    ],
    name='gis'
)
>>> gis.ads.feature_type = ['gis']
>>> gis.ads.feature_domain()
constraints: []
stats:
  count: 13
  missing: 3
  unique: 10
values: GIS

```

**Returns**

Domain based on the GIS feature type.

**Return type**

ads.feature\_engineering.schema.Domain

**static feature\_plot**(*x: Series*) → Axes

Shows the location of given address on map base on longitude and latitude.

**Examples**

```

>>> gis = pd.Series([
    "69.196241,-125.017615",
    "5.2272595,-143.465712",
    "-33.9855425,-153.445155",
    "43.340610,86.460554",
    "24.2811855,-162.380403",
    "2.7849025,-7.328156",
    "45.033805,157.490179",
    "-1.818319,-80.681214",
    "-44.510428,-169.269477",
    "-56.3344375,-166.407038",
    "",
    np.NaN,
    None
],
    name='gis'
)
>>> gis.ads.feature_type = ['gis']
>>> gis.ads.feature_plot()

```

**Returns**

Plot object for the series based on the GIS feature type.

**Return type**

matplotlib.axes.\_subplots.AxesSubplot

**static feature\_stat**(*x: Series*) → DataFrame

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

### Examples

```
>>> gis = pd.Series([
    "69.196241,-125.017615",
    "5.2272595,-143.465712",
    "-33.9855425,-153.445155",
    "43.340610,86.460554",
    "24.2811855,-162.380403",
    "2.7849025,-7.328156",
    "45.033805,157.490179",
    "-1.818319,-80.681214",
    "-44.510428,-169.269477",
    "-56.3344375,-166.407038",
    "",
    np.NaN,
    None
],
    name='gis'
)
>>> gis.ads.feature_type = ['gis']
>>> gis.ads.feature_stat()
      gis
count   13
unique   10
missing   3
```

### Returns

Summary statistics of the Series provided.

### Return type

pandas.DataFrame

**validator =**

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning =**

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

ads.feature\_engineering.feature\_type.gis.default\_handler(*data: Series, \*args, \*\*kwargs*) → Series

Processes given data and indicates if the data matches requirements.

### Parameters

**data** (pandas.Series) – The data to process.

### Returns

The logical list indicating if the data matches requirements.

### Return type

pandas.Series

#### 18.1.1.11.24 ads.feature\_engineering.feature\_type.integer module

The module that represents an Integer feature type.

##### Classes:

###### **Integer**

The Integer feature type.

**class** ads.feature\_engineering.feature\_type.integer.**Integer**

Bases: *FeatureType*

Type representing integer values.

##### **description**

The feature type description.

###### **Type**

str

##### **name**

The feature type name.

###### **Type**

str

##### **warning**

Provides functionality to register warnings and invoke them.

###### **Type**

*FeatureWarning*

##### **validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x: pd.Series*) → pd.DataFrame

Generates feature statistics.

**feature\_plot**(*x: pd.Series*) → plt.Axes

Shows distributions of datasets using box plot.

**description** = 'Type representing integer values.'

**classmethod feature\_domain**(*x: Series*) → Domain

Generate the domain of the data of this feature type.

##### **Examples**

```
>>> s = pd.Series([True, False, True, False, np.NaN, None], name='integer')
>>> s.ads.feature_type = ['integer']
>>> s.ads.feature_domain()
constraints: []
stats:
  count: 6
  freq: 2
  missing: 2
  top: true
```

(continues on next page)

(continued from previous page)

```
unique: 2
values: Integer
```

**Returns**

Domain based on the Integer feature type.

**Return type**

`ads.feature_engineering.schema.Domain`

**static** `feature_plot(x: Series) → Axes`

Shows distributions of datasets using box plot.

**Examples**

```
>>> x = pd.Series([1, 0, 1, 2, 3, 4, np.nan], name='integer')
>>> x.ads.feature_type = ['integer']
>>> x.ads.feature_plot()
```

**Returns**

Plot object for the series based on the Integer feature type.

**Return type**

`matplotlib.axes._subplots.AxesSubplot`

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, mean, standard deviation, sample minimum, lower quartile, median, 75%, upper quartile, max and missing(count) if there is any.

**Examples**

```
>>> x = pd.Series([1, 0, 1, 2, 3, 4, np.nan], name='integer')
>>> x.ads.feature_type = ['integer']
>>> x.ads.feature_stat()
   Metric      Value
0    count         7
1    mean          1
2  standard deviation    1
3  sample minimum      0
4   lower quartile      1
5    median          1
6   upper quartile      2
7  sample maximum      4
8    missing         1
```

**Returns**

Summary statistics of the Series or Dataframe provided.

**Return type**

`pandas.DataFrame`

```
validator =  
<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator  
object>  
  
warning =  
<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>
```

#### 18.1.1.11.25 ads.feature\_engineering.feature\_type.ip\_address module

The module that represents an IPAddress feature type.

##### Classes:

###### IPAddress

The IPAddress feature type.

**class** ads.feature\_engineering.feature\_type.ip\_address.IPAddress

Bases: *FeatureType*

Type representing IP Address.

###### description

The feature type description.

###### Type

str

###### name

The feature type name.

###### Type

str

###### warning

Provides functionality to register warnings and invoke them.

###### Type

*FeatureWarning*

###### validator

Provides functionality to register validators and invoke them.

**feature\_stat**(*x*: *pd.Series*) → *pd.DataFrame*

Generates feature statistics.

##### Example

```
>>> from ads.feature_engineering.feature_type.ip_address import IPAddress  
>>> import pandas as pd  
>>> import numpy as np  
>>> s = pd.Series(['192.168.0.1', '2001:db8::', '', np.NaN, None], name='ip_address'  
↪ )  
>>> s.ads.feature_type = ['ip_address']  
>>> IPAddress.validator.is_ip_address(s)  
0      True  
1      True
```

(continues on next page)

(continued from previous page)

```

2    False
3    False
4    False
Name: ip_address, dtype: bool

```

**description** = 'Type representing IP Address.'

**classmethod** `feature_domain(x: Series) → Domain`

Generate the domain of the data of this feature type.

### Examples

```

>>> s = pd.Series(['2002:db8::', '192.168.0.1', '2001:db8::', '2002:db8::', np.
↳NaN, None], name='ip_address')
>>> s.ads.feature_type = ['ip_address']
>>> s.ads.feature_domain()
constraints: []
stats:
  count: 6
  missing: 2
  unique: 3
values: IPAddress

```

#### Returns

Domain based on the IPAddress feature type.

#### Return type

`ads.feature_engineering.schema.Domain`

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

### Examples

```

>>> s = pd.Series(['2002:db8::', '192.168.0.1', '2001:db8::', '2002:db8::', np.
↳NaN, None], name='ip_address')
>>> s.ads.feature_type = ['ip_address']
>>> s.ads.feature_stat()
Metric Value
0      count    6
1     unique    2
2     missing    2

```

#### Returns

Summary statistics of the Series provided.

#### Return type

`pandas.DataFrame`

```
validator =  
<ads.feature_engineering.feature_type.handler.feature_validator.FeatureValidator  
object>  
  
warning =  
<ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning object>
```

```
ads.feature_engineering.feature_type.ip_address.default_handler(data: Series, *args, **kwargs)  
→ Series
```

Processes given data and indicates if the data matches requirements.

**Parameters**

**data** (`pandas.Series`) – The data to process.

**Returns**

The logical list indicating if the data matches requirements.

**Return type**

`pandas.Series`

#### 18.1.1.11.26 `ads.feature_engineering.feature_type.ip_address_v4` module

The module that represents an `IpAddressV4` feature type.

**Classes:**

**`IpAddressV4`**

The `IpAddressV4` feature type.

```
class ads.feature_engineering.feature_type.ip_address_v4.IpAddressV4
```

Bases: *FeatureType*

Type representing IP Address V4.

**description**

The feature type description.

**Type**

`str`

**name**

The feature type name.

**Type**

`str`

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

```
feature_stat(x: pd.Series) → pd.DataFrame
```

Generates feature statistics.



### Example

```
>>> from ads.feature_engineering.feature_type.ip_address_v4 import IPAddressV4
>>> import pandas as pd
>>> import numpy as np
>>> s = pd.Series(['192.168.0.1', '2001:db8::', '', np.NaN, None], name='ip_address')
>>> s.ads.feature_type = ['ip_address_v4']
>>> IPAddressV4.validator.is_ip_address_v4(s)
0      True
1     False
2     False
3     False
4     False
Name: ip_address, dtype: bool
```

**description** = 'Type representing IP Address V4.'

**classmethod** `feature_domain(x: Series) → Domain`

Generate the domain of the data of this feature type.

### Examples

```
>>> s = pd.Series(['192.168.0.1', '192.168.0.2', '192.168.0.3', '192.168.0.4', np.NaN, None], name='ip_address_v4')
>>> s.ads.feature_type = ['ip_address_v4']
>>> s.ads.feature_domain()
constraints: []
stats:
  count: 6
  missing: 2
  unique: 4
values: IPAddressV4
```

#### Returns

Domain based on the IPAddressV4 feature type.

#### Return type

`ads.feature_engineering.schema.Domain`

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

## Examples

```
>>> s = pd.Series(['192.168.0.1', '192.168.0.2', '192.168.0.3', '192.168.0.4',
↳ np.NaN, None], name='ip_address')
>>> s.ads.feature_type = ['ip_address_v4']
>>> s.ads.feature_stat()
      Metric  Value
0         count    6
1         unique    4
2         missing    2
```

### Returns

Summary statistics of the Series provided.

### Return type

pandas.DataFrame

**validator =**

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning =**

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

ads.feature\_engineering.feature\_type.ip\_address\_v4.**default\_handler**(data: Series, \*args, \*\*kwargs) → Series

Processes given data and indicates if the data matches requirements.

### Parameters

**data** (pandas.Series) – The data to process.

### Returns

The logical list indicating if the data matches requirements.

### Return type

pandas.Series

## 18.1.1.11.27 ads.feature\_engineering.feature\_type.ip\_address\_v6 module

The module that represents an IPAddressV6 feature type.

### Classes:

#### IPAddressV6

The IPAddressV6 feature type.

**class** ads.feature\_engineering.feature\_type.ip\_address\_v6.**IPAddressV6**

Bases: *FeatureType*

Type representing IP Address V6.

### description

The feature type description.

### Type

str

**name**

The feature type name.

**Type**

str

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x*: *pd.Series*) → *pd.DataFrame*

Generates feature statistics.

**Example**

```
>>> from ads.feature_engineering.feature_type.ip_address_v6 import IpAddressV6
>>> import pandas as pd
>>> import numpy as np
>>> s = pd.Series(['192.168.0.1', '2001:db8::', '', np.NaN, None], name='ip_address_v6')
>>> s.ads.feature_type = ['ip_address_v6']
>>> IpAddressV6.validator.is_ip_address_v6(s)
0    False
1     True
2    False
3    False
4    False
Name: ip_address, dtype: bool
```

**description** = 'Type representing IP Address V6.'

**classmethod feature\_domain**(*x*: *Series*) → *Domain*

Generate the domain of the data of this feature type.

**Examples**

```
>>> s = pd.Series(['2002:db8::', '2001:db8::', '2001:db8::', '2002:db8::', np.NaN, None], name='ip_address_v6')
>>> s.ads.feature_type = ['ip_address_v6']
>>> s.ads.feature_domain()
constraints: []
stats:
  count: 6
  missing: 2
  unique: 2
values: IpAddressV6
```

**Returns**

Domain based on the IPAddressV6 feature type.

**Return type**

ads.feature\_engineering.schema.Domain

**static** **feature\_stat**(*x: Series*) → DataFrame

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

**Examples**

```
>>> s = pd.Series(['2002:db8::', '2001:db8::', '2001:db8::', '2002:db8::', np.
↳NaN, None], name='ip_address')
>>> s.ads.feature_type = ['ip_address_v6']
>>> s.ads.feature_stat()
Metric Value
0      count    6
1     unique    2
2     missing    2
```

**Returns**

Summary statistics of the Series provided.

**Return type**

Pandas Dataframe

**validator** =

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning** =

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

ads.feature\_engineering.feature\_type.ip\_address\_v6.**default\_handler**(*data: Series, \*args, \*\*kwargs*) → Series

Processes given data and indicates if the data matches requirements.

**Parameters**

**data** (pandas.Series) – The data to process.

**Returns**

The logical list indicating if the data matches requirements.

**Return type**

pandas.Series

### 18.1.1.11.28 ads.feature\_engineering.feature\_type.lat\_long module

The module that represents a LatLong feature type.

#### Classes:

##### **LatLong**

The LatLong feature type.

#### Functions:

##### **default\_handler(data: pd.Series) -> pd.Series**

Processes given data and indicates if the data matches requirements.

##### **class ads.feature\_engineering.feature\_type.lat\_long.LatLong**

Bases: *String*

Type representing longitude and latitude.

##### **description**

The feature type description.

##### **Type**

str

##### **name**

The feature type name.

##### **Type**

str

##### **warning**

Provides functionality to register warnings and invoke them.

##### **Type**

*FeatureWarning*

##### **validator**

Provides functionality to register validators and invoke them.

##### **feature\_stat(x: pd.Series) -> pd.DataFrame**

Generates feature statistics.

##### **feature\_plot(x: pd.Series) -> plt.Axes**

Shows the location of given address on map base on longitude and latitude.

#### Example

```

>>> from ads.feature_engineering.feature_type.lat_long import LatLong
>>> import pandas as pd
>>> s = pd.Series(["-18.2193965, -93.587285",
                  "-21.0255305, -122.478584",
                  "85.103913, 19.405744",
                  "82.913736, 178.225672",
                  "62.9795085, -66.989705",
                  "54.5604395, 95.235090",
                  "24.2811855, -162.380403",
                  "-1.818319, -80.681214",

```

(continues on next page)

(continued from previous page)

```

        None,
        "(51.816119, 175.979008)",
        "(54.3392995,-11.801615)"],
        name='latlong')
>>> s.ads.feature_type = ['latlong']
>>> LatLong.validator.is_lat_long(s)
0      True
1      True
2      True
3      True
4      True
5      True
6      True
7      True
8      False
9      True
10     True
Name: latlong, dtype: bool

```

**description** = 'Type representing longitude and latitude.'

**classmethod** **feature\_domain**(*x: Series*) → Domain

Generate the domain of the data of this feature type.

## Examples

```

>>> latlong_series = pd.Series([
    "69.196241,-125.017615",
    "5.2272595,-143.465712",
    "-33.9855425,-153.445155",
    "43.340610,86.460554",
    "24.2811855,-162.380403",
    "2.7849025,-7.328156",
    "45.033805,157.490179",
    "-1.818319,-80.681214",
    "-44.510428,-169.269477",
    "-56.3344375,-166.407038",
    "",
    np.NaN,
    None
],
    name='latlong')
>>> latlong_series.ads.feature_type = ['latlong']
>>> latlong_series.ads.feature_domain()
constraints: []
stats:
  count: 13
  missing: 3
  unique: 10
values: LatLong

```

**Returns**

Domain based on the LatLong feature type.

**Return type**

ads.feature\_engineering.schema.Domain

**static feature\_plot**(*x: Series*) → Axes

Shows the location of given address on map base on longitude and latitude.

**Examples**

```
>>> latlong_series = pd.Series([
    "69.196241,-125.017615",
    "5.2272595,-143.465712",
    "-33.9855425,-153.445155",
    "43.340610,86.460554",
    "24.2811855,-162.380403",
    "2.7849025,-7.328156",
    "45.033805,157.490179",
    "-1.818319,-80.681214",
    "-44.510428,-169.269477",
    "-56.3344375,-166.407038",
    "",
    np.NaN,
    None
],
    name='latlong'
)
>>> latlong_series.ads.feature_type = ['lat_long']
>>> latlong_series.ads.feature_plot()
```

**Returns**

Plot object for the series based on the LatLong feature type.

**Return type**

matplotlib.axes.\_subplots.AxesSubplot

**static feature\_stat**(*x: Series*) → DataFrame

Generate feature statistics.

Feature statistics include (total)count, unique(count) and missing(count) if there is any.

**Examples**

```
>>> latlong_series = pd.Series([
    "69.196241,-125.017615",
    "5.2272595,-143.465712",
    "-33.9855425,-153.445155",
    "43.340610,86.460554",
    "24.2811855,-162.380403",
    "2.7849025,-7.328156",
    "45.033805,157.490179",
    "-1.818319,-80.681214",
    "-44.510428,-169.269477",
    "-56.3344375,-166.407038",
    "",
    np.NaN,
    None
])
>>> latlong_series.ads.feature_stat()
```

(continues on next page)

(continued from previous page)

```

        "-1.818319,-80.681214",
        "-44.510428,-169.269477",
        "-56.3344375,-166.407038",
        "",
        np.NaN,
        None
    ],
    name='latlong'
)
>>> latlong_series.ads.feature_type = ['lat_long']
>>> latlong_series.ads.feature_stat()
Metric Value
0      count    13
1     unique    10
2     missing     3

```

**Returns**

Summary statistics of the Series or Dataframe provided.

**Return type**

pandas.DataFrame

**validator =**

&lt;ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object&gt;

**warning =**

&lt;ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object&gt;

ads.feature\_engineering.feature\_type.lat\_long.default\_handler(*data: Series, \*args, \*\*kwargs*) → Series

Processes given data and indicates if the data matches requirements.

**Parameters****data** (pandas.Series) – The data to process.**Returns**

The logical list indicating if the data matches requirements.

**Return type**

pandas.Series

**18.1.1.11.29 ads.feature\_engineering.feature\_type.object module**

The module that represents an Object feature type.

**Classes:****Object**

The Object feature type.

**class** ads.feature\_engineering.feature\_type.object.ObjectBases: *FeatureType*

Type representing object.



**description**

The feature type description.

**Type**

str

**name**

The feature type name.

**Type**

str

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

**description** = 'Type representing object.'

**classmethod** **feature\_domain()**

**Returns**

Nothing.

**Return type**

None

**validator** =

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning** =

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

### 18.1.1.11.30 ads.feature\_engineering.feature\_type.ordinal module

The module that represents an Ordinal feature type.

**Classes:****Ordinal**

The Ordinal feature type.

**class** ads.feature\_engineering.feature\_type.ordinal.**Ordinal**

Bases: *FeatureType*

Type representing ordered values.

**description**

The feature type description.

**Type**

str

**name**

The feature type name.

**Type**

str

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x*: *pd.Series*) → *pd.DataFrame*

Generates feature statistics.

**feature\_plot**(*x*: *pd.Series*) → *plt.Axes*

Shows the counts of observations in each categorical bin using bar chart.

**description** = 'Type representing ordered values.'

**classmethod feature\_domain**(*x*: *Series*) → *Domain*

Generate the domain of the data of this feature type.

**Examples**

```
>>> x = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, np.nan], name='ordinal')
>>> x.ads.feature_type = ['ordinal']
>>> x.ads.feature_domain()
constraints:
- expression: $x in [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
  language: python
stats:
  count: 10
  missing: 1
  unique: 9
values: Ordinal
```

**Returns**

Domain based on the Ordinal feature type.

**Return type**

*ads.feature\_engineering.schema.Domain*

**static feature\_plot**(*x*: *Series*) → *Axes*

Shows the counts of observations in each categorical bin using bar chart.

## Examples

```
>>> x = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, np.nan], name='ordinal')
>>> x.ads.feature_type = ['ordinal']
>>> x.ads.feature_plot()
```

### Returns

The bar chart plot object for the series based on the Continuous feature type.

### Return type

matplotlib.axes.\_subplots.AxesSubplot

**static** `feature_stat(x: Series) → DataFrame`

Generates feature statistics.

Feature statistics include (total)count, unique(count), and missing(count) if there is any.

## Examples

```
>>> x = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, np.nan], name='ordinal')
>>> x.ads.feature_type = ['ordinal']
>>> x.ads.feature_stat()
Metric  Value
0      count  10
1     unique   9
2    missing   1
```

### Returns

Summary statistics of the Series or Dataframe provided.

### Return type

pandas.DataFrame

**validator =**

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning =**

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

### 18.1.1.11.31 ads.feature\_engineering.feature\_type.phone\_number module

The module that represents a Phone Number feature type.

#### Classes:

##### PhoneNumber

The Phone Number feature type.

#### Functions:

**default\_handler(data: pd.Series) -> pd.Series**

Processes given data and indicates if the data matches requirements.

**class** `ads.feature_engineering.feature_type.phone_number.PhoneNumber`

Bases: *String*

Type representing phone numbers.

**description**

The feature type description.

**Type**

`str`

**name**

The feature type name.

**Type**

`str`

**warning**

Provides functionality to register warnings and invoke them.

**Type**

*FeatureWarning*

**validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x*: *pd.Series*) → *pd.DataFrame*

Generates feature statistics.

## Examples

```
>>> from ads.feature_engineering.feature_type.phone_number import PhoneNumber
>>> import pandas as pd
>>> s = pd.Series([None, "1-640-124-5367", "1-573-916-4412"])
>>> PhoneNumber.validator.is_phone_number(s)
0    False
1     True
2     True
dtype: bool
```

**description** = 'Type representing phone numbers.'

**classmethod** **feature\_domain**(*x*: *Series*) → *Domain*

Generate the domain of the data of this feature type.

## Examples

```
>>> s = pd.Series(['2068866666', '6508866666', '2068866666', '', np.NaN, np.nan,
↪ None], name='phone')
>>> s.ads.feature_type = ['phone_number']
>>> s.ads.feature_domain()
constraints: []
stats:
  count: 7
```

(continues on next page)

(continued from previous page)

```

missing: 4
unique: 2
values: PhoneNumber

```

**Returns**

Domain based on the PhoneNumber feature type.

**Return type**

ads.feature\_engineering.schema.Domain

**static feature\_stat**(*x: Series*) → DataFrame

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count) if there is any.

**Examples**

```

>>> s = pd.Series(['2068866666', '6508866666', '2068866666', '', np.NaN, np.nan,
→ None], name='phone')
>>> s.ads.feature_type = ['phone_number']
>>> s.ads.feature_stat()
Metric  Value
1      count   7
2      unique   2
3      missing  4

```

**Returns**

Summary statistics of the Series or Dataframe provided.

**Return type**

pandas.DataFrame

**validator =**

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning =**

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

ads.feature\_engineering.feature\_type.phone\_number.default\_handler(*data: Series, \*args, \*\*kwargs*) → Series

Processes given data and indicates if the data matches requirements.

**Parameters**

**data** (pandas.Series) – The data to process.

**Returns**

The logical list indicating if the data matches requirements.

**Return type**

pandas.Series

### 18.1.1.11.32 ads.feature\_engineering.feature\_type.string module

The module that represents a String feature type.

#### Classes:

##### **String**

The feature type that represents string values.

**class** ads.feature\_engineering.feature\_type.string.**String**

Bases: *FeatureType*

Type representing string values.

#### **description**

The feature type description.

##### **Type**

str

#### **name**

The feature type name.

##### **Type**

str

#### **warning**

Provides functionality to register warnings and invoke them.

##### **Type**

*FeatureWarning*

#### **validator**

Provides functionality to register validators and invoke them.

**feature\_stat**(*x: pd.Series*) → pd.DataFrame

Generates feature statistics.

**feature\_plot**(*x: pd.Series*) → plt.Axes

Shows distributions of datasets using wordcloud.

#### **Example**

```
>>> from ads.feature_engineering.feature_type.string import String
>>> import pandas as pd
>>> s = pd.Series(["Hello", "world", None], name='string')
>>> String.validator.is_string(s)
0      True
1      True
2     False
Name: string, dtype: bool
```

**description** = 'Type representing string values.'

**classmethod** **feature\_domain**(*x: Series*) → Domain

Generate the domain of the data of this feature type.

## Examples

```
>>> string = pd.Series(['S', 'C', 'S', 'S', 'S', 'Q', 'S', 'S', 'S', 'C', 'S',
↳ 'S', 'S',
    'S', 'S', 'S', 'Q', 'S', 'S', '', np.NaN, None], name='string')
>>> string.ads.feature_type = ['string']
>>> string.ads.feature_domain()
constraints: []
stats:
  count: 22
  missing: 3
  unique: 3
values: String
```

### Returns

Domain based on the String feature type.

### Return type

ads.feature\_engineering.schema.Domain

**static feature\_plot**(*x: Series*) → Axes

Shows distributions of datasets using wordcloud.

## Examples

```
>>> string = pd.Series(['S', 'C', 'S', 'S', 'S', 'Q', 'S', 'S', 'S', 'C', 'S',
↳ 'S', 'S',
    'S', 'S', 'S', 'Q', 'S', 'S', '', np.NaN, None], name='string')
>>> string.ads.feature_type = ['string']
>>> string.ads.feature_plot()
```

### Returns

Plot object for the series based on the String feature type.

### Return type

matplotlib.axes.\_subplots.AxesSubplot

**static feature\_stat**(*x: Series*) → DataFrame

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count) if there is any.

## Examples

```
>>> string = pd.Series(['S', 'C', 'S', 'S', 'S', 'Q', 'S', 'S', 'S', 'C', 'S',
↳ 'S', 'S',
    'S', 'S', 'S', 'Q', 'S', 'S', '', np.NaN, None], name='string')
>>> string.ads.feature_type = ['string']
>>> string.ads.feature_stat()
Metric Value
0      count    22
```

(continues on next page)

(continued from previous page)

1	unique	3
2	missing	3

**Returns**

Summary statistics of the Series or Dataframe provided.

**Return type**

Pandas Dataframe

**validator =**

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning =**

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

ads.feature\_engineering.feature\_type.string.default\_handler(*data: Series, \*args, \*\*kwargs*) → Series

Processes given data and indicates if the data matches requirements.

**Parameters**

**data** (*pd.Series*) – The data to process.

**Returns**

pd.Series

**Return type**

The logical list indicating if the data matches requirements.

### 18.1.1.11.33 ads.feature\_engineering.feature\_type.text module

The module that represents a Text feature type.

**Classes:****Text**

The Text feature type.

**class** ads.feature\_engineering.feature\_type.text.Text

Bases: *String*

Type representing text values.

**description**

The feature type description.

**Type**

str

**name**

The feature type name.

**Type**

str





#### 18.1.1.11.34 ads.feature\_engineering.feature\_type.unknown module

The module that represents an Unknown feature type.

##### Classes:

###### Text

The Unknown feature type.

**class** ads.feature\_engineering.feature\_type.unknown.**Unknown**

Bases: *FeatureType*

Type representing third-party dtypes.

###### description

The feature type description.

###### Type

str

###### name

The feature type name.

###### Type

str

###### warning

Provides functionality to register warnings and invoke them.

###### Type

*FeatureWarning*

###### validator

Provides functionality to register validators and invoke them.

**description** = 'Type representing unknown type.'

**classmethod** feature\_domain()

###### Returns

Nothing.

###### Return type

None

**validator** =

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning** =

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

### 18.1.1.11.35 ads.feature\_engineering.feature\_type.zip\_code module

The module that represents a ZipCode feature type.

#### Classes:

##### **ZipCode**

The ZipCode feature type.

#### Functions:

##### **default\_handler(data: pd.Series) -> pd.Series**

Processes given data and indicates if the data matches requirements.

##### **class ads.feature\_engineering.feature\_type.zip\_code.ZipCode**

Bases: *String*

Type representing postal code.

##### **description**

The feature type description.

##### **Type**

str

##### **name**

The feature type name.

##### **Type**

str

##### **warning**

Provides functionality to register warnings and invoke them.

##### **Type**

*FeatureWarning*

##### **validator**

Provides functionality to register validators and invoke them.

##### **feature\_stat(x: pd.Series) → pd.DataFrame**

Generates feature statistics.

##### **feature\_plot(x: pd.Series) → plt.Axes**

Shows the geometry distribution base on location of zipcode.

#### Example

```
>>> from ads.feature_engineering.feature_type.zip_code import ZipCode
>>> import pandas as pd
>>> import numpy as np
>>> s = pd.Series(["94065", "90210", np.NaN, None], name='zipcode')
>>> ZipCode.validator.is_zip_code(s)
0    True
1    True
2   False
3   False
Name: zipcode, dtype: bool
```

**description** = 'Type representing postal code.'

**classmethod** **feature\_domain**(*x*: *Series*) → *Domain*

Generate the domain of the data of this feature type.

### Examples

```
>>> zipcode = pd.Series([94065, 90210, np.NaN, None], name='zipcode')
>>> zipcode.ads.feature_type = ['zip_code']
>>> zipcode.ads.feature_domain()
constraints: []
stats:
  count: 4
  missing: 2
  unique: 2
values: ZipCode
```

#### Returns

Domain based on the ZipCode feature type.

#### Return type

ads.feature\_engineering.schema.Domain

**static** **feature\_plot**(*x*: *Series*) → *Axes*

Shows the geometry distribution base on location of zipcode.

### Examples

```
>>> zipcode = pd.Series([94065, 90210, np.NaN, None], name='zipcode')
>>> zipcode.ads.feature_type = ['zip_code']
>>> zipcode.ads.feature_plot()
```

#### Returns

Plot object for the series based on the ZipCode feature type.

#### Return type

matplotlib.axes.\_subplots.AxesSubplot

**static** **feature\_stat**(*x*: *Series*) → *DataFrame*

Generates feature statistics.

Feature statistics include (total)count, unique(count) and missing(count).

## Examples

```
>>> zipcode = pd.Series([94065, 90210, np.NaN, None], name='zipcode')
>>> zipcode.ads.feature_type = ['zip_code']
>>> zipcode.ads.feature_stat()
Metric Value
0      count    4
1     unique    2
2     missing    2
```

### Returns

Summary statistics of the Series provided.

### Return type

Pandas Dataframe

**validator =**

<ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator object>

**warning =**

<ads.feature\_engineering.feature\_type.handler.feature\_warning.FeatureWarning object>

ads.feature\_engineering.feature\_type.zip\_code.default\_handler(*data: Series, \*args, \*\*kwargs*) → Series

Processes given data and indicates if the data matches requirements.

### Parameters

**data** (*pd.Series*) – The data to process.

### Returns

**pd.Series**

### Return type

The logical list indicating if the data matches requirements.

## 18.1.1.11.36 ads.feature\_engineering.feature\_type.handler.feature\_validator module

The module that helps to register custom validators for the feature types and extending registered validators with dispatching based on the specific arguments.

## Classes

### FeatureValidator

The Feature Validator class to manage custom validators.

### FeatureValidatorMethod

The Feature Validator Method class. Extends methods which requires dispatching based on the specific arguments.

**class** ads.feature\_engineering.feature\_type.handler.feature\_validator.FeatureValidator

Bases: object

The Feature Validator class to manage custom validators.

**register**(*self*, *name*: str, *handler*: Callable, *condition*: Union[Tuple, Dict[str, Any]] = None, *replace*: bool = False) → None

Registers new validator.

**unregister**(*self*, *name*: str, *condition*: Union[Tuple, Dict[str, Any]] = None) → None

Unregisters validator.

**registered**(*self*) → pd.DataFrame

Gets the list of registered validators.

## Examples

```
>>> series = pd.Series(['+1-202-555-0141', '+1-202-555-0142'], name='Phone Number')
```

```
>>> def phone_number_validator(data: pd.Series) -> pd.Series:
...     print("phone_number_validator")
...     return data
```

```
>>> def universal_phone_number_validator(data: pd.Series, country_code) -> pd.
↳Series:
...     print("universal_phone_number_validator")
...     return data
```

```
>>> def us_phone_number_validator(data: pd.Series, country_code) -> pd.Series:
...     print("us_phone_number_validator")
...     return data
```

```
>>> PhoneNumber.validator.register(name="is_phone_number", handler=phone_number_
↳validator, replace=True)
>>> PhoneNumber.validator.register(name="is_phone_number", handler=universal_phone_
↳number_validator, condition = ('country_code',))
>>> PhoneNumber.validator.register(name="is_phone_number", handler=us_phone_number_
↳validator, condition = {'country_code': '+1'})
```

```
>>> PhoneNumber.validator.is_phone_number(series)
phone_number_validator
0      +1-202-555-0141
1      +1-202-555-0142
```

```
>>> PhoneNumber.validator.is_phone_number(series, country_code = '+7')
universal_phone_number_validator
0      +1-202-555-0141
1      +1-202-555-0142
```

```
>>> PhoneNumber.validator.is_phone_number(series, country_code = '+1')
us_phone_number_validator
0      +1-202-555-0141
1      +1-202-555-0142
```

```
>>> PhoneNumber.validator.registered()
Validator Condition
↪Handler
-----
↪-
0 is_phone_number () phone_number_
↪validator
1 is_phone_number ('country_code') universal_phone_number_
↪validator
2 is_phone_number {'country_code': '+1'} us_phone_number_
↪validator
```

```
>>> series.ads.validator.is_phone_number()
phone_number_validator
0 +1-202-555-0141
1 +1-202-555-0142
```

```
>>> series.ads.validator.is_phone_number(country_code = '+7')
universal_phone_number_validator
0 +1-202-555-0141
1 +1-202-555-0142
```

```
>>> series.ads.validator.is_phone_number(country_code = '+1')
us_phone_number_validator
0 +1-202-555-0141
1 +1-202-555-0142
```

Initializes the FeatureValidator.

**register**(*name*: str, *handler*: Callable, *condition*: Optional[Union[Tuple, Dict[str, Any]]] = None, *replace*: bool = False) → None

Registers new validator.

#### Parameters

- **name** (str) – The validator name.
- **handler** (callable) – The handler.
- **condition** (Union[Tuple, Dict[str, Any]]) – The condition for the validator.
- **replace** (bool) – The flag indicating if the registered validator should be replaced with the new one.

#### Returns

Nothing.

#### Return type

None

#### Raises

- **ValueError** – The name is empty or handler is not provided.
- **TypeError** – The handler is not callable. The name of the validator is not a string.
- **ValidatorAlreadyExists** – The validator is already registered.

**registered()** → DataFrame

Gets the list of registered validators.

**Returns**

The list of registered validators.

**Return type**

pd.DataFrame

**unregister**(*name: str, condition: Optional[Union[Tuple, Dict[str, Any]]] = None*) → None

Unregisters validator.

**Parameters**

- **name** (*str*) – The name of the validator to be unregistered.
- **condition** (*Union[Tuple, Dict[str, Any]]*) – The condition for the validator to be unregistered.

**Returns**

Nothing.

**Return type**

None

**Raises**

- **TypeError** – The name of the validator is not a string.
- **ValidatorNotFound** – The validator not found.
- **ValidatorWithConditionNotFound** – The validator with provided condition not found.

**class** ads.feature\_engineering.feature\_type.handler.feature\_validator.**FeatureValidatorMethod**(*handler: Callable*)

Bases: object

The Feature Validator Method class.

Extends methods which requires dispatching based on the specific arguments.

**register**(*self, condition: Union[Tuple, Dict[str, Any]], handler: Callable*) → None

Registers new handler.

**unregister**(*self, condition: Union[Tuple, Dict[str, Any]]*) → None

Unregisters existing handler.

**registered**(*self*) → pd.DataFrame

Gets the list of registered handlers.

Initializes the Feature Validator Method.

**Parameters**

**handler** (*Callable*) – The handler that will be called by default if suitable one not found.

**register**(*condition: Union[Tuple, Dict[str, Any]], handler: Callable*) → None

Registers new handler.

**Parameters**

- **condition** (*Union[Tuple, Dict[str, Any]]*) – The condition which will be used to register a new handler.
- **handler** (*Callable*) – The handler to be registered.



**Returns**

Nothing.

**Return type**

None

**Raises****ValueError** – If condition not provided or provided in the wrong format. If handler not provided or has wrong format.**registered()** → DataFrame

Gets the list of registered handlers.

**Returns**

The list of registerd handlers.

**Return type**

pd.DataFrame

**unregister**(*condition: Union[Tuple, Dict[str, Any]]*) → None

Unregisters existing handler.

**Parameters****condition** (*Union[Tuple, Dict[str, Any]]*) – The condition which will be used to unregister a handler.**Returns**

Nothing.

**Return type**

None

**Raises****ValueError** – If condition not provided or provided in the wrong format. If condition not registered.**exception** `ads.feature_engineering.feature_type.handler.feature_validator.ValidatorAlreadyExists`(*name: str*)

Bases: ValueError

**exception** `ads.feature_engineering.feature_type.handler.feature_validator.ValidatorNotFound`(*name: str*)

Bases: ValueError

**exception** `ads.feature_engineering.feature_type.handler.feature_validator.ValidatorWithConditionAlreadyE`

Bases: ValueError

**exception** `ads.feature_engineering.feature_type.handler.feature_validator.ValidatorWithConditionNotFound`

Bases: ValueError

**exception** `ads.feature_engineering.feature_type.handler.feature_validator.WrongHandlerMethodSignature`(*han**str;*  
*con*  
*di-*  
*tion*  
*str;*  
*han*  
*dle*  
*str*

Bases: ValueError

### 18.1.1.11.37 ads.feature\_engineering.feature\_type.handler.feature\_warning module

The module that helps to register custom warnings for the feature types.

## Classes

### FeatureWarning

The Feature Warning class. Provides functionality to register warning handlers and invoke them.

## Examples

```
>>> warning = FeatureWarning()
>>> def warning_handler_zeros_count(data):
...     return pd.DataFrame(
...         [['Zeros', 'Age has 38 zeros', 'Count', 38]],
...         columns=['Warning', 'Message', 'Metric', 'Value'])
>>> def warning_handler_zeros_percentage(data):
...     return pd.DataFrame(
...         [['Zeros', 'Age has 12.2% zeros', 'Percentage', '12.2%']],
...         columns=['Warning', 'Message', 'Metric', 'Value'])
>>> warning.register(name="zeros_count", handler=warning_handler_zeros_count)
>>> warning.register(name="zeros_percentage", handler=warning_handler_percentage)
>>> warning.registered()
```

	Name	Handler
0	zeros_count	warning_handler_zeros_count
1	zeros_percentage	warning_handler_zeros_percentage

```
>>> warning.zeros_percentage(data_series)
```

	Warning	Message	Metric	Value
0	Zeros	Age has 38 zeros	Count	38

```
>>> warning.zeros_count(data_series)
```

	Warning	Message	Metric	Value
1	Zeros	Age has 12.2% zeros	Percentage	12.2%

```
>>> warning(data_series)
```

	Warning	Message	Metric	Value
0	Zeros	Age has 38 zeros	Count	38
1	Zeros	Age has 12.2% zeros	Percentage	12.2%

```
>>> warning.unregister('zeros_count')
>>> warning(data_series)
```

(continues on next page)

(continued from previous page)

	Warning	Message	Metric	Value
-----				
0	Zeros	Age has 12.2% zeros	Percentage	12.2%

**class** `ads.feature_engineering.feature_type.handler.feature_warning.FeatureWarning`

Bases: `object`

The Feature Warning class.

Provides functionality to register warning handlers and invoke them.

**register**(*self*, *name*: *str*, *handler*: *Callable*) → *None*

Registers a new warning for the feature type.

**unregister**(*self*, *name*: *str*) → *None*

Unregisters warning.

**registered**(*self*) → `pd.DataFrame`

Gets the list of registered warnings.

## Examples

```
>>> warning = FeatureWarning()
>>> def warning_handler_zeros_count(data):
...     return pd.DataFrame(
...         [['Zeros', 'Age has 38 zeros', 'Count', 38]],
...         columns=['Warning', 'Message', 'Metric', 'Value'])
>>> def warning_handler_zeros_percentage(data):
...     return pd.DataFrame(
...         [['Zeros', 'Age has 12.2% zeros', 'Percentage', '12.2%']],
...         columns=['Warning', 'Message', 'Metric', 'Value'])
>>> warning.register(name="zeros_count", handler=warning_handler_zeros_count)
>>> warning.register(name="zeros_percentage", handler=warning_handler_percentage)
>>> warning.registered()
```

	Warning	Handler
-----		
0	zeros_count	warning_handler_zeros_count
1	zeros_percentage	warning_handler_zeros_percentage

```
>>> warning.zeros_percentage(data_series)
```

	Warning	Message	Metric	Value
-----				
0	Zeros	Age has 38 zeros	Count	38

```
>>> warning.zeros_count(data_series)
```

	Warning	Message	Metric	Value
-----				
1	Zeros	Age has 12.2% zeros	Percentage	12.2%

```
>>> warning(data_series)
```

	Warning	Message	Metric	Value
-----				

(continues on next page)

(continued from previous page)

0	Zeros	Age has 38 zeros	Count	38
1	Zeros	Age has 12.2% zeros	Percentage	12.2%

```
>>> warning.unregister('zeros_count')
>>> warning(data_series)
```

	Warning	Message	Metric	Value
	-----	-----	-----	-----
0	Zeros	Age has 12.2% zeros	Percentage	12.2%

Initializes the FeatureWarning.

**register**(*name: str, handler: Callable, replace: bool = False*) → None

Registers a new warning.

#### Parameters

- **name** (*str*) – The warning name.
- **handler** (*callable*) – The handler associated with the warning.
- **replace** (*bool*) – The flag indicating if the registered warning should be replaced with the new one.

#### Returns

Nothing

#### Return type

None

#### Raises

- **ValueError** – If warning name is empty or handler not defined.
- **TypeError** – If handler is not callable.
- **WarningAlreadyExists** – If warning is already registered.

**registered**() → DataFrame

Gets the list of registered warnings.

#### Return type

pd.DataFrame

### Examples

```
>>> The list of registered warnings in DataFrame format.
```

	Name	Handler
	-----	-----
0	zeros_count	warning_handler_zeros_count
1	zeros_percentage	warning_handler_zeros_percentage

**unregister**(*name: str*) → None

Unregisters warning.

#### Parameters

- **name** (*str*) – The name of warning to be unregistered.

**Returns**

Nothing.

**Return type**

None

**Raises**

- **ValueError** – If warning name is not provided or empty.
- **WarningNotFound** – If warning not found.

**18.1.1.11.38 ads.feature\_engineering.feature\_type.handler.warnings module**

The module with all default warnings provided to user. These are registered to relevant feature types directly in the feature type files themselves.

`ads.feature_engineering.feature_type.handler.warnings.high_cardinality_handler(s: Series) → DataFrame`

Warning if number of unique values (including Nan) in series is greater than or equal to 15.

**Parameters**

**s** (*pd.Series*) – Pandas series - column of some feature type.

**Returns**

Dataframe with 4 columns 'Warning', 'Message', 'Metric', 'Value' and 1 rows, which lists count of unique values.

**Return type**

pd.DataFrame

`ads.feature_engineering.feature_type.handler.warnings.missing_values_handler(s: Series) → DataFrame`

Warning for > 5 percent missing values (Nans) in series.

**Parameters**

**s** (*pd.Series*) – Pandas series - column of some feature type.

**Returns**

Dataframe with 4 columns 'Warning', 'Message', 'Metric', 'Value' and 2 rows, where first row is count of missing values and second is percentage of missing values.

**Return type**

pd.DataFrame

`ads.feature_engineering.feature_type.handler.warnings.skew_handler(s: Series) → DataFrame`

Warning if absolute value of skew is greater than 1.

**Parameters**

**s** (*pd.Series*) – Pandas series - column of some feature type, expects continuous values.

**Returns**

Dataframe with 4 columns 'Warning', 'Message', 'Metric', 'Value' and 1 rows, which lists skew value of that column.

**Return type**

pd.DataFrame

`ads.feature_engineering.feature_type.handler.warnings.zeros_handler(s: Series) → DataFrame`

Warning for greater than 10 percent zeros in series.

**Parameters**

**s** (*pd.Series*) – Pandas series - column of some feature type.

**Returns**

Dataframe with 4 columns ‘Warning’, ‘Message’, ‘Metric’, ‘Value’ and 2 rows, where first row is count of zero values and second is percentage of zero values.

**Return type**

pd.DataFrame

### 18.1.1.11.39 Module contents

### 18.1.1.12 ads.hpo package

#### 18.1.1.12.1 Submodules

#### 18.1.1.12.2 ads.hpo.distributions module

**class** `ads.hpo.distributions.CategoricalDistribution`(*choices: Sequence[Union[None, bool, int, float, str]]*)

Bases: *Distribution*

A categorical distribution.

**Parameters**

**choices** – Parameter value candidates. It is recommended to restrict the types of the choices to the following: None, bool, int, float and str.

**class** `ads.hpo.distributions.DiscreteUniformDistribution`(*low: float, high: float, step: float*)

Bases: *Distribution*

A discretized uniform distribution in the linear domain.

---

**Note:** If the range [low, high] is not divisible by *q*, high will be replaced with the maximum of  $kq + \text{lowhigh}$ , where *k* is an integer.

---

**Parameters**

- **low** (*float*) – Lower endpoint of the range of the distribution. *low* is included in the range.
- **high** (*float*) – Upper endpoint of the range of the distribution. *high* is included in the range.
- **step** (*float*) – A discretization step.

**class** `ads.hpo.distributions.Distribution`(*dist*)

Bases: object

Defines the abstract base class for hyperparameter search distributions

**get\_distribution()**

Returns the distribution

```
class ads.hpo.distributions.DistributionEncode(*, skipkeys=False, ensure_ascii=True,
                                             check_circular=True, allow_nan=True,
                                             sort_keys=False, indent=None, separators=None,
                                             default=None)
```

Bases: `JSONEncoder`

Constructor for `JSONEncoder`, with sensible defaults.

If `skipkeys` is false, then it is a `TypeError` to attempt encoding of keys that are not `str`, `int`, `float` or `None`. If `skipkeys` is `True`, such items are simply skipped.

If `ensure_ascii` is true, the output is guaranteed to be `str` objects with all incoming non-ASCII characters escaped. If `ensure_ascii` is false, the output can contain non-ASCII characters.

If `check_circular` is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an `OverflowError`). Otherwise, no such check takes place.

If `allow_nan` is true, then `NaN`, `Infinity`, and `-Infinity` will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a `ValueError` to encode such floats.

If `sort_keys` is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. `None` is the most compact representation.

If specified, `separators` should be an (item\_separator, key\_separator) tuple. The default is `(' ', ': ')` if `indent` is `None` and `('', ': ')` otherwise. To get the most compact JSON representation, you should specify `(';', ':')` to eliminate whitespace.

If specified, `default` is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

**default**(*dist*: [Distribution](#)) → `Dict[str, Any]`

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement `default` like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

**static from\_json**(*json\_object*: `Dict[Any, Any]`)

```
class ads.hpo.distributions.IntLogUniformDistribution(low: float, high: float, step: float = 1)
```

Bases: [Distribution](#)

A uniform distribution on integers in the log domain.

#### Parameters

- **low** – Lower endpoint of the range of the distribution. *low* is included in the range.

- **high** – Upper endpoint of the range of the distribution. *high* is included in the range.
- **step** – A step for spacing between values.

**class** `ads.hpo.distributions.IntUniformDistribution`(*low: float, high: float, step: float = 1*)

Bases: *Distribution*

A uniform distribution on integers.

---

**Note:** If the range  $[low, high]$  is not divisible by *step*, *high* will be replaced with the maximum of  $k \times step + low$ , where  $k$  is an integer.

---

#### Parameters

- **low** – Lower endpoint of the range of the distribution. *low* is included in the range.
- **high** – Upper endpoint of the range of the distribution. *high* is included in the range.
- **step** – A step for spacing between values.

**class** `ads.hpo.distributions.LogUniformDistribution`(*low: float, high: float*)

Bases: *Distribution*

A uniform distribution in the log domain.

#### Parameters

- **low** – Lower endpoint of the range of the distribution. *low* is included in the range.
- **high** – Upper endpoint of the range of the distribution. *high* is excluded from the range.

**class** `ads.hpo.distributions.UniformDistribution`(*low: float, high: float*)

Bases: *Distribution*

A uniform distribution in the linear domain.

#### Parameters

- **low** – Lower endpoint of the range of the distribution. *low* is included in the range.
- **high** – Upper endpoint of the range of the distribution. *high* is excluded from the range.

`ads.hpo.distributions.decode`(*s: str*)

Decodes a string to an object

#### Parameters

**s** (*str*) – The string being decoded to a distribution object

#### Returns

Decoded string

#### Return type

*Distribution* or Dict

`ads.hpo.distributions.encode`(*o: Distribution*) → str

Encodes a distribution to a string

#### Parameters

**o** (*Distribution*) – The distribution to encode

#### Returns

The distribution encoded as a string



**Return type**str (*DistributionEncode*)**18.1.1.12.3 ads.hpo.search\_cv module**

```
class ads.hpo.search_cv.ADSTuner(model, strategy='perfunctory', scoring=None, cv=5, study_name=None,
                                storage=None, load_if_exists=True, random_state=None, loglevel=20,
                                n_jobs=1, X=None, y=None)
```

Bases: BaseEstimator

Hyperparameter search with cross-validation.

Returns a hyperparameter tuning object

**Parameters**

- **model** – Object to use to fit the data. This is assumed to implement the scikit-learn estimator or pipeline interface.
- **strategy** – *perfunctory*, *detailed* or a dictionary/mapping of hyperparameter and its distribution . If obj:*perfunctory*, picks a few relatively more important hyperparameters to tune . If obj:*detailed*, extends to a larger search space. If obj:dict, user defined search space: Dictionary where keys are hyperparameters and values are distributions. Distributions are assumed to implement the ads distribution interface.
- **scoring** (*Optional[Union[Callable[...], str]]*) – String or callable to evaluate the predictions on the validation data. If *None*, score on the estimator is used.
- **cv** (*int*) – Integer to specify the number of folds in a CV splitter. If estimator is a classifier and y is either binary or multiclass, `sklearn.model_selection.StratifiedKFold` is used. otherwise, `sklearn.model_selection.KFold` is used.
- **study\_name** (*str*,) – Name of the current experiment for the ADSTuner object. One ADSTuner object can only be attached to one study\_name.
- **storage** – Database URL. (e.g. `sqlite:///example.db`). Default to `sqlite:///tmp/hpo_*.db`.
- **load\_if\_exists** – Flag to control the behavior to handle a conflict of study names. In the case where a study named `study_name` already exists in the `storage`, a *DuplicatedStudyError* is raised if `load_if_exists` is set to *False*. Otherwise, the existing one is returned.
- **random\_state** – Seed of the pseudo random number generator. If *int*, this is the seed used by the random number generator. If *None*, the global random state from `numpy.random` is used.
- **loglevel** – loglevel. can be `logging.NOTSET`, `logging.INFO`, `logging.DEBUG`, `logging.WARNING`
- **n\_jobs** (*int*) – Number of parallel jobs. -1 means using all processors.
- **X** (*TwoDimArrayLikeType, Union[List[List[float]], np.ndarray*,) –
- **pd.DataFrame** – Training data.
- **spmatrix** – Training data.
- **ADSDData**] – Training data.
- **y** (*Union[OneDimArrayLikeType, TwoDimArrayLikeType], optional*) –
- **OneDimArrayLikeType** (*Union[List[float], np.ndarray, pd.Series]*) –

- **TwoDimArrayLikeType** (`Union[List[List[float]], np.ndarray, pd.DataFrame, spmatrix, ADSDData]`) – Target.

Example:

```
from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.svm import SVC

tuner = ADSTuner(
    SVC(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)

X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)])
```

**property best\_index**

returns: Index which corresponds to the best candidate parameter setting. :rtype: int

**property best\_params**

returns: Parameters of the best trial. :rtype: Dict[str, Any]

**property best\_score**

returns: Mean cross-validated score of the best estimator. :rtype: float

**best\_scores**(*n*: int = 5, *reverse*: bool = True)

Return the best scores from the study

**Parameters**

- **n** (int) – The maximum number of results to show. Defaults to 5. If *None* or negative return all.
- **reverse** (bool) – Whether to reverse the sort order so results are in descending order. Defaults to *True*

**Returns**

List of the best scores

**Return type**

list[float or int]

**Raises**

**ValueError** –

**get\_status**()

return the status of the current tuning process.

Alias for the property *status*.

**Returns**

The status of the process

**Return type**

Status

Example:

```

from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier

tuner = ADSTuner(
    SGDClassifier(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)
tuner.search_space({'max_iter': 100})
X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)])
tuner.get_status()

```

**halt()**

Halt the current running tuning process.

**Returns**

Nothing

**Return type**

None

**Raises**

*InvalidStateTransition* –

Example:

```

from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier

tuner = ADSTuner(
    SGDClassifier(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)
tuner.search_space({'max_iter': 100})
X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)])
tuner.halt()

```

**is\_completed()****Returns**

*True* if the *ADSTuner* instance has completed; *False* otherwise.

**Return type**

bool

**is\_halted()**

**Returns**

*True* if the [ADSTuner](#) instance is halted; *False* otherwise.

**Return type**

bool

**is\_running()****Returns**

*True* if the [ADSTuner](#) instance is running; *False* otherwise.

**Return type**

bool

**is\_terminated()****Returns**

*True* if the [ADSTuner](#) instance has been terminated; *False* otherwise.

**Return type**

bool

**property n\_trials**

returns: Number of completed trials. Alias for *trial\_count*. :rtype: int

**static optimizer**(*study\_name*, *pruner*, *sampler*, *storage*, *load\_if\_exists*, *objective\_func*, *global\_start*, *global\_stop*, *\*\*kwargs*)

Static method for running ADSTuner tuning process

**Parameters**

- **study\_name** (*str*) – The name of the study.
- **pruner** – The pruning method for pruning trials.
- **sampler** – The sampling method used for tuning.
- **storage** (*str*) – Storage endpoint.
- **load\_if\_exists** (*bool*) – Load existing study if it exists.
- **objective\_func** – The objective function to be maximized.
- **global\_start** (*multiprocessing.Value*) – The global start time.
- **global\_stop** (*multiprocessing.Value*) – The global stop time.
- **kwargs** (*dict*) – Keyword/value pairs passed into the optimize process

**Raises**

**Exception** – Raised for any exceptions thrown by the underlying optimization process

**Returns**

Nothing

**Return type**

None

**plot\_best\_scores**(*best=True*, *inferior=True*, *time\_interval=1*, *fig\_size=(800, 500)*)

Plot optimization history of all trials in a study.

**Parameters**

- **best** – controls whether to plot the lines for the best scores so far.
- **inferior** – controls whether to plot the dots for the actual objective scores.

- **time\_interval** – how often(in seconds) the plot refresh to check on the new trial results.
- **fig\_size** (*tuple*) – width and height of the figure.

**Returns**

Nothing.

**Return type**

None

**plot\_contour\_scores**(*params=None, time\_interval=1, fig\_size=(800, 500)*)

Contour plot of the scores.

**Parameters**

- **params** (*Optional[List[str]]*) – Parameter list to visualize. Defaults to all.
- **time\_interval** (*float*) – Time interval for the plot. Defaults to 1.
- **fig\_size** (*tuple[int, int]*) – Figure size. Defaults to (800, 500).

**Returns**

Nothing.

**Return type**

None

**plot\_edf\_scores**(*time\_interval=1, fig\_size=(800, 500)*)

Plot the EDF (empirical distribution function) of the scores.

Only completed trials are used.

**Parameters**

- **time\_interval** (*float*) – Time interval for the plot. Defaults to 1.
- **fig\_size** (*tuple[int, int]*) – Figure size. Defaults to (800, 500).

**Returns**

Nothing.

**Return type**

None

**plot\_intermediate\_scores**(*time\_interval=1, fig\_size=(800, 500)*)

Plot intermediate values of all trials in a study.

**Parameters**

- **time\_interval** (*float*) – Time interval for the plot. Defaults to 1.
- **fig\_size** (*tuple[int, int]*) – Figure size. Defaults to (800, 500).

**Returns**

Nothing.

**Return type**

None

**plot\_parallel\_coordinate\_scores**(*params=None, time\_interval=1, fig\_size=(800, 500)*)

Plot the high-dimentional parameter relationships in a study.

Note that, If a parameter contains missing values, a trial with missing values is not plotted.

**Parameters**

- **params** (*Optional[List[str]]*) – Parameter list to visualize. Defaults to all.
- **time\_interval** (*float*) – Time interval for the plot. Defaults to 1.
- **fig\_size** (*tuple[int, int]*) – Figure size. Defaults to (800, 500).

**Returns**

Nothing.

**Return type**

None

**plot\_param\_importance**(*importance\_evaluator='Fanova', time\_interval=1, fig\_size=(800, 500)*)

Plot hyperparameter importances.

**Parameters**

- **importance\_evaluator** (*str*) – Importance evaluator. Valid values: “Fanova”, “Mean-DecreaseImpurity”. Defaults to “Fanova”.
- **time\_interval** (*float*) – How often the plot refresh to check on the new trial results.
- **fig\_size** (*tuple*) – Width and height of the figure.

**Raises**

**NotImplementedError** – Raised for unsupported importance evaluators

**Returns**

Nothing.

**Return type**

None

**resume()**

Resume the current halted tuning process.

**Returns**

Nothing

**Return type**

None

Example:

```
from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier

tuner = ADSTuner(
    SGDClassifier(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)
tuner.search_space({'max_iter': 100})
X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)])
tuner.halt()
tuner.resume()
```

**property score\_remaining**

returns: The difference between the best score and the optimal score. :rtype: float

**Raises**

**ExitCriterionError** – Error is raised if there is no score-based criteria for tuning.

**property scoring\_name**

returns: Scoring name. :rtype: str

**search\_space**(*strategy=None, overwrite=False*)

Returns the search space. If strategy is not passed in, return the existing search space. When strategy is passed in, overwrite the existing search space if overwrite is set True, otherwise, only update the existing search space.

**Parameters**

- **strategy** (*Union[str, dict], optional*) – perfunctory, detailed or a dictionary/mapping of the hyperparameters and their distributions. If obj:*perfunctory*, picks a few relatively more important hyperparameters to tune. If obj:*detailed*, extends to a larger search space. If obj:*dict*, user defined search space: Dictionary where keys are parameters and values are distributions. Distributions are assumed to implement the ads distribution interface.
- **overwrite** (*bool, optional*) – Ignored when strategy is None. Otherwise, search space is overwritten if overwrite is set True and updated if it is False.

**Returns**

A mapping of the hyperparameters and their distributions.

**Return type**

dict

Example:

```
from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier

tuner = ADSTuner(
    SGDClassifier(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)
tuner.search_space({'max_iter': 100})
X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)])
tuner.search_space()
```

**property sklearn\_steps**

returns: Search space which corresponds to the best candidate parameter setting. :rtype: int

**property status**

returns: The status of the current tuning process. :rtype: Status

**terminate()**

Terminate the current tuning process.

**Returns**

Nothing

**Return type**

None

Example:

```
from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier

tuner = ADSTuner(
    SGDClassifier(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)
tuner.search_space({'max_iter': 100})
X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)])
tuner.terminate()
```

**property time\_elapsed**

Return the time in seconds that the HPO process has been searching

**Returns****int****Return type**

The number of seconds the HPO process has been searching

**property time\_remaining**

Returns the number of seconds remaining in the study

**Returns****int****Return type**

Number of seconds remaining in the budget. 0 if complete/terminated

**Raises*****ExitCriterionError*** – Error is raised if time has not been included in the budget.**property time\_since\_resume**

Return the seconds since the process has been resumed from a halt.

**Returns****int****Return type**

the number of seconds since the process was last resumed

**Raises*****NoRestartError*** –**property trial\_count**returns: Number of completed trials. Alias for *trial\_count*. :rtype: int



**property trials**

returns: Trial data up to this point. :rtype: pandas.DataFrame

**trials\_export**(*file\_uri*, *metadata*=None, *script\_dict*={'model': None, 'scoring': None})

Export the meta data as well as files needed to reconstruct the ADSTuner object to the object storage. Data is not stored. To resume the same ADSTuner object from object storage and continue tuning from previous trials, you have to provide the dataset.

**Parameters**

- **file\_uri** (*str*) – Object storage path, 'oci://bucketname@namespace/filepath/on/objectstorage'. For example, `oci://test_bucket@ociodscust/tuner/test.zip`
- **metadata** (*str*, *optional*) – User defined metadata
- **script\_dict** (*dict*, *optional*) – Script paths for model and scoring. This is only recommended for unsupported models and user-defined scoring functions. You can store the model and scoring function in a dictionary with keys *model* and *scoring* and the respective paths as values. The model and scoring scripts must import necessary libraries for the script to run. The *model* and *scoring* variables must be set to your model and scoring function.

**Returns**

Nothing

**Return type**

None

Example:

```
# Print out a list of supported models
from ads.hpo.ads_search_space import model_list
print(model_list)

# Example scoring dictionary
{'model': '/home/datascience/advanced-ds/notebooks/scratch/ADSTunerV2/mymodel.py',
 'scoring': '/home/datascience/advanced-ds/notebooks/scratch/ADSTunerV2/
customized_scoring.py'}
```

Example:

```
from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier

tuner = ADSTuner(
    SGDClassifier(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)
tuner.search_space({'max_iter': 100})
X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)], synchronous=True)
tuner.trials_export('oci://<bucket_name>@<namespace>/tuner/test.zip')
```

**classmethod** `trials_import(file_uri, delete_zip_file=True, target_file_path=None)`

Import the database file from the object storage

#### Parameters

- **file\_uri** (*str*) – ‘oci://bucketname@namespace/filepath/on/objectstorage’ Example: ‘oci://<bucket\_name>@<namespace>/tuner/test.zip’
- **delete\_zip\_file** (*bool*, defaults to *True*, optional) – Whether delete the zip file afterwards.
- **target\_file\_path** (*str*, optional) – The path where the zip file will be saved. For example, ‘/home/datascience/myfile.zip’.

#### Returns

ADSTuner object

#### Return type

*ADSTuner*

### Examples

```
>>> from ads.hpo.stopping_criterion import *
>>> from ads.hpo.search_cv import ADSTuner
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import SGDClassifier
>>> X, y = load_iris(return_X_y=True)
>>> tuner = ADSTuner.trials_import('oci://<bucket_name><namespace>/tuner/test.
↳zip')
>>> tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)], synchronous=True)
```

#### property `trials_remaining`

returns: The number of trials remaining in the budget. :rtype: int

#### Raises

***ExitCriterionError*** – Raised if the current tuner does not include a trials-based exit condition.

**tune**(*X=None, y=None, exit\_criterion=[], loglevel=None, synchronous=False*)

Run hyperparameter tuning until one of the `exit_criterion` is met. The default is to run 50 trials.

#### Parameters

- **X** (*TwoDimArrayLikeType*, *Union[List[List[float]], np.ndarray, pd.DataFrame, spmatrix, ADSData]*) – Training data.
- **y** (*Union[OneDimArrayLikeType, TwoDimArrayLikeType]*, optional) –
- **OneDimArrayLikeType** (*Union[List[float], np.ndarray, pd.Series]*) –
- **TwoDimArrayLikeType** (*Union[List[List[float]], np.ndarray, pd.DataFrame, spmatrix, ADSData]*) – Target.
- **exit\_criterion** (*list*, optional) – A list of ads stopping criterion. Can be *ScoreValue()*, *NTrials()*, *TimeBudget()*. For example, [*ScoreValue*(0.96), *NTrials*(40), *TimeBudget*(10)]. It will exit when any of the stopping criterion is satisfied in the *exit\_criterion* list. By default, the run will stop after 50 trials.
- **loglevel** (*int*, optional) – Log level.

- **synchronous** (*boolean, optional*) – Tune synchronously or not. Defaults to *False*

**Returns**

Nothing

**Return type**

None

Example:

```
from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.svm import SVC

tuner = ADSTuner(
    SVC(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)
tuner.search_space({'max_iter': 100})
X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)])
```

**wait()**

Wait for the current tuning process to finish running.

**Returns**

Nothing

**Return type**

None

Example:

```
from ads.hpo.stopping_criterion import *
from ads.hpo.search_cv import ADSTuner
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier

tuner = ADSTuner(
    SGDClassifier(),
    strategy='detailed',
    scoring='f1_weighted',
    random_state=42
)
tuner.search_space({'max_iter': 100})
X, y = load_iris(return_X_y=True)
tuner.tune(X=X, y=y, exit_criterion=[TimeBudget(1)])
tuner.wait()
```

**exception ads.hpo.search\_cv.DuplicatedStudyError**

Bases: Exception

*DuplicatedStudyError* is raised when a new tuner process is created with a study name that already exists in storage.

**exception** `ads.hpo.search_cv.ExitCriterionError`

Bases: `Exception`

*ExitCriterionError* is raised when an attempt is made to check exit status for a different exit type than the tuner was initialized with. For example, if an HPO study has an exit criteria based on the number of trials and a request is made for the time remaining, which is a different exit criterion, an exception is raised.

**exception** `ads.hpo.search_cv.InvalidStateTransition`

Bases: `Exception`

*Invalid State Transition* is raised when an invalid transition request is made, such as calling `halt` without a running process.

**exception** `ads.hpo.search_cv.NoRestartError`

Bases: `Exception`

*NoRestartError* is raised when an attempt is made to check how many seconds have transpired since the HPO process was last resumed from a halt. This can happen if the process has been terminated or it was never halted and then resumed to begin with.

**class** `ads.hpo.search_cv.State(value)`

Bases: `Enum`

An enumeration.

**COMPLETED** = 5

**HALTED** = 3

**INITIATED** = 1

**RUNNING** = 2

**TERMINATED** = 4

#### 18.1.1.12.4 `ads.hpo.stopping_criterion`

**class** `ads.hpo.stopping_criterion.NTrials(n_trials: int)`

Bases: `object`

Exit based on number of trials.

**Parameters**

**n\_trials** (*int*) – Number of trials (sets of hyperparameters tested). If `None`, there is no limitation on the number of trials.

**Returns**

`NTrials` object

**Return type**

*NTrials*

**class** `ads.hpo.stopping_criterion.ScoreValue(score: float)`

Bases: `object`

Exit if the score is greater than or equal to the threshold.

**Parameters**

**score** (*float*) – The threshold for exiting the tuning process. If a trial value is greater or equal to *score*, process exits.

**Returns**

ScoreValue object

**Return type***ScoreValue***class** `ads.hpo.stopping_criterion.TimeBudget`(*seconds: float*)

Bases: object

Exit based on the number of seconds.

**Parameters****seconds** (*float*) – Time limit, in seconds. If None there is no time limit.**Returns**

TimeBudget object

**Return type***TimeBudget***18.1.1.12.5 Module contents****18.1.1.13 ads.jobs package****18.1.1.13.1 Submodules****18.1.1.13.2 ads.jobs.ads\_job module****class** `ads.jobs.ads_job.Job`(*name: Optional[str] = None, infrastructure=None, runtime=None*)

Bases: Builder

Represents a Job containing infrastructure and runtime.

**Example**

Here is an example for creating and running a job:

```

from ads.jobs import Job, DataScienceJob, PythonRuntime
# Define an OCI Data Science job to run a python script
job = (
    Job(name="<job_name>")
    .with_infrastructure(
        DataScienceJob()
        .with_compartment_id("<compartment_ocid>")
        .with_project_id("<project_ocid>")
        .with_subnet_id("<subnet_ocid>")
        .with_shape_name("VM.Standard.E3.Flex")
        .with_shape_config_details(memory_in_gbs=16, ocpus=1)
        .with_block_storage_size(50)
        .with_log_group_id("<log_group_ocid>")
        .with_log_id("<log_ocid>")
    )
    .with_runtime(
        ScriptRuntime()

```

(continues on next page)

(continued from previous page)

```

        .with_source("oci://bucket_name@namespace/path/to/script.py")
        .with_service_conda("tensorflow26_p37_cpu_v2")
        .with_environment_variable(ENV="value")
        .with_argument("argument", key="value")
        .with_freeform_tag(tag_name="tag_value")
    )
)
# Create and Run the job
run = job.create().run()
# Stream the job run outputs
run.watch()

```

If you are in an OCI notebook session and you would like to use the same infrastructure configurations, the infrastructure configuration can be simplified. Here is another example of creating and running a jupyter notebook as a job:

```

from ads.jobs import Job, DataScienceJob, NotebookRuntime
# Define an OCI Data Science job to run a jupyter Python notebook
job = (
    Job(name="<job_name>")
    .with_infrastructure(
        # The same configurations as the OCI notebook session will be used.
        DataScienceJob()
        .with_log_group_id("<log_group_ocid>")
        .with_log_id("<log_ocid>")
    )
    .with_runtime(
        NotebookRuntime()
        .with_notebook("path/to/notebook.ipynb")
        .with_service_conda(tensorflow26_p37_cpu_v2")
        # Saves the notebook with outputs to OCI object storage.
        .with_output("oci://bucket_name@namespace/path/to/dir")
    )
).create()
# Run and monitor the job
run = job.run().watch()
# Download the notebook and outputs to local directory
run.download(to_dir="path/to/local/dir/")

```

See also:

**https**

[//docs.oracle.com/en-us/iaas/tools/ads-sdk/latest/user\\_guide/jobs/index.html](https://docs.oracle.com/en-us/iaas/tools/ads-sdk/latest/user_guide/jobs/index.html)

Initializes a job.

**The infrastructure and runtime can be configured when initializing the job,**  
or by calling `with_infrastructure()` and `with_runtime()`.

The infrastructure should be a subclass of ADS job Infrastructure, e.g., `DataScienceJob`, `DataFlow`. The runtime should be a subclass of ADS job Runtime, e.g., `PythonRuntime`, `ScriptRuntime`.

### Parameters

- **name** (*str*, *optional*) – The name of the job, by default None. If it is None, a default name may be generated by the infrastructure, depending on the implementation of the infrastructure. For OCI data science job, the default name contains the job artifact name and a timestamp. If no artifact, a randomly generated easy to remember name with timestamp will be generated, like ‘strange-spider-2022-08-17-23:55.02’.
- **infrastructure** (*Infrastructure*, *optional*) – Job infrastructure, by default None
- **runtime** (*Runtime*, *optional*) – Job runtime, by default None.

**create**(\*\**kwargs*) → *Job*

Creates the job on the infrastructure.

**Returns**

The job instance (self)

**Return type**

*Job*

**static dataflow\_job**(*compartment\_id: Optional[str] = None*, \*\**kwargs*) → List[*Job*]

List data flow jobs under a given compartment.

**Parameters**

- **compartment\_id** (*str*) – compartment id
- **kwargs** – additional keyword arguments

**Returns**

list of Job instances

**Return type**

List[*Job*]

**static datascience\_job**(*compartment\_id: Optional[str] = None*, \*\**kwargs*) → List[*DataScienceJob*]

Lists the existing data science jobs in the compartment.

**Parameters**

**compartment\_id** (*str*) – The compartment ID for listing the jobs. This is optional if running in an OCI notebook session. The jobs in the same compartment of the notebook session will be returned.

**Returns**

A list of Job objects.

**Return type**

list

**delete**() → None

Deletes the job from the infrastructure.

**download**(*to\_dir: str*, *output\_uri=None*, \*\**storage\_options*)

Downloads files from remote output URI to local.

**Parameters**

- **to\_dir** (*str*) – Local directory to which the files will be downloaded to.
- **output\_uri** ((*str*, *optional*). *Default is None.*) – The remote URI from which the files will be downloaded. Defaults to None. If output\_uri is not specified, this method will try to get the output\_uri from the runtime.

- **storage\_options** – Extra keyword arguments for particular storage connection. This method uses fsspec to download the files from remote URI. storage\_options will to be passed into fsspec.open\_files().

**Returns**

The job instance (self)

**Return type**

*Job*

**Raises**

**AttributeError** – The output\_uri is not specified and the runtime is not configured with output\_uri.

**static from\_dataflow\_job**(job\_id: str) → *Job*

Create a Data Flow job given a job id.

**Parameters**

**job\_id** (str) – id of the job

**Returns**

a Job instance

**Return type**

*Job*

**static from\_datascience\_job**(job\_id) → *Job*

Loads a data science job from OCI.

**Parameters**

**job\_id** (str) – OCID of an existing data science job.

**Returns**

A job instance.

**Return type**

*Job*

**classmethod from\_dict**(config: dict) → *Job*

Initializes a job from a dictionary containing the configurations.

**Parameters**

**config** (dict) – A dictionary containing the infrastructure and runtime specifications.

**Returns**

A job instance

**Return type**

*Job*

**Raises**

**NotImplementedError** – If the type of the infrastructure or runtime is not supported.

**property id: str**

The ID of the job. For jobs running on OCI, this is the OCID.

**Returns**

ID of the job.

**Return type**

str



**property infrastructure:** Union[[DataScienceJob](#), [DataFlow](#)]

The job infrastructure.

**Returns**

Job infrastructure.

**Return type**

Infrastructure

**property kind:** str

The kind of the object as showing in YAML.

**Returns**

“job”

**Return type**

str

**property name:** str

The name of the job. For jobs running on OCI, this is the display name.

**Returns**

The name of the job.

**Return type**

str

**run**(*name=None, args=None, env\_var=None, freeform\_tags=None, wait=False*) → Union[[DataScienceJobRun](#), [DataFlowRun](#)]

Runs the job.

**Parameters**

- **name** (*str, optional*) – Name of the job run, by default None. The infrastructure handles the naming of the job run. For data science job, if a name is not provided, a default name will be generated containing the job name and the timestamp of the run. If no artifact, a randomly generated easy to remember name with timestamp will be generated, like ‘strange-spider-2022-08-17-23:55.02’.
- **args** (*str, optional*) – Command line arguments for the job run, by default None. This will override the configurations on the job. If this is None, the args from the job configuration will be used.
- **env\_var** (*dict, optional*) – Additional environment variables for the job run, by default None
- **freeform\_tags** (*dict, optional*) – Freeform tags for the job run, by default None
- **wait** (*bool, optional*) – Indicate if this method call should wait for the job run. By default False, this method returns as soon as the job run is created. If this is set to True, this method will stream the job logs and wait until it finishes, similar to *job.run().watch()*.

**Returns**

A job run instance, depending on the infrastructure.

**Return type**

Job Run Instance

**run\_list**(*\*\*kwargs*) → list

Gets a list of runs of the job.

**Returns**

A list of job run instances, the actual object type depends on the infrastructure.

**Return type**

list

**property runtime: Runtime**

The job runtime.

**Returns**

The job runtime

**Return type**

Runtime

**status()** → str

Status of the job

**Returns**

Status of the job

**Return type**

str

**to\_dict()** → dict

Serialize the job specifications to a dictionary.

**Returns**

A dictionary containing job specifications.

**Return type**

dict

**with\_infrastructure(infrastructure)** → *Job*

Sets the infrastructure for the job.

**Parameters**

**infrastructure** (*Infrastructure*) – Job infrastructure.

**Returns**

The job instance (self)

**Return type**

*Job*

**with\_name(name: str)** → *Job*

Sets the job name.

**Parameters**

**name** (*str*) – Job name.

**Returns**

The job instance (self)

**Return type**

*Job*

**with\_runtime(runtime)** → *Job*

Sets the runtime for the job.

**Parameters**

**runtime** (*Runtime*) – Job runtime.

**Returns**

The job instance (self)

**Return type**

*Job*

**18.1.1.13.3 ads.jobs.builders.runtimes.python\_runtime module**

```
class ads.jobs.builders.runtimes.python_runtime.CondaRuntime(spec: Optional[Dict] = None,  
                                                           **kwargs)
```

Bases: Runtime

Represents a job runtime with conda pack

Initialize the object with specifications.

User can either pass in the specification as a dictionary or through keyword arguments.

**Parameters**

- **spec** (*dict*, *optional*) – Object specification, by default None
- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

```
CONST_CONDA = 'conda'
```

```
CONST_CONDA_REGION = 'region'
```

```
CONST_CONDA_SLUG = 'slug'
```

```
CONST_CONDA_TYPE = 'type'
```

```
CONST_CONDA_TYPE_CUSTOM = 'published'
```

```
CONST_CONDA_TYPE_SERVICE = 'service'
```

```
CONST_CONDA_URI = 'uri'
```

```
attribute_map = {'conda': 'conda', 'env': 'env', 'freeformTags': 'freeform_tags'}
```

```
property conda: dict
```

The conda pack specification

**Returns**

A dictionary with “type” and “slug” as keys.

**Return type**

dict

```
with_custom_conda(uri: str, region: Optional[str] = None)
```

Specifies the custom conda pack for running the job

**Parameters**

- **uri** (*str*) – The OCI object storage URI for the conda pack, e.g. “oci://your\_bucket@namespace/object\_name.” In the Environment Explorer of an OCI notebook session, this is shown as the “source” of the conda pack.

- **region** (*str*, *optional*) – The region of the bucket storing the custom conda pack, by default None. If region is not specified, ADS will use the region from your authentication credentials, \* For API Key, config[“region”] is used. \* For Resource Principal, signer.region is used.

This is required if the conda pack is stored in a different region.

**Returns**

The runtime instance.

**Return type**

self

**See also:****https**

[//docs.oracle.com/en-us/iaas/data-science/using/conda\\_publishs\\_object.htm](https://docs.oracle.com/en-us/iaas/data-science/using/conda_publishs_object.htm)

**with\_service\_conda**(*slug: str*)

Specifies the service conda pack for running the job

**Parameters**

**slug** (*str*) – The slug name of the service conda pack

**Returns**

The runtime instance.

**Return type**

self

```
class ads.jobs.builders.runtimes.python_runtime.DataFlowNotebookRuntime(spec: Optional[Dict]
                                                                    = None, **kwargs)
```

Bases: [DataFlowRuntime](#), [NotebookRuntime](#)

Initialize the object with specifications.

User can either pass in the specification as a dictionary or through keyword arguments.

**Parameters**

- **spec** (*dict*, *optional*) – Object specification, by default None
- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

**convert** (*overwrite=False*)

```
class ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime(spec: Optional[Dict] = None,
                                                                **kwargs)
```

Bases: [CondaRuntime](#)

Initialize the object with specifications.

User can either pass in the specification as a dictionary or through keyword arguments.

**Parameters**

- **spec** (*dict*, *optional*) – Object specification, by default None
- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

```

CONST_ARCHIVE_BUCKET = 'archiveBucket'

CONST_ARCHIVE_URI = 'archiveUri'

CONST_CONDA_AUTH_TYPE = 'condaAuthType'

CONST_CONFIGURATION = 'configuration'

CONST_SCRIPT_BUCKET = 'scriptBucket'

CONST_SCRIPT_PATH = 'scriptPathURI'

property archive_bucket: str
    Bucket to save archive zip

property archive_uri
    The Uri of archive zip

attribute_map = {'archiveUri': 'archive_uri', 'condaAuthType': 'conda_auth_type',
'configuration': 'configuration', 'env': 'env', 'freeformTags': 'freeform_tags',
'scriptBucket': 'script_bucket', 'scriptPathURI': 'script_path_uri'}

property configuration: dict
    Configuration for Spark

convert(**kwargs)

property script_bucket: str
    Bucket to save script

property script_uri: str
    The URI of the source code

with_archive_bucket(bucket) → DataFlowRuntime
    Set object storage bucket to save the archive zip, in case archive uri given is local.

    Parameters
        bucket (str) – name of the bucket

    Returns
        runtime instance itself

    Return type
        DataFlowRuntime

with_archive_uri(uri: str) → DataFlowRuntime
    Set archive uri (which is a zip file containing dependencies).

    Parameters
        uri (str) – uri to the archive zip

    Returns
        runtime instance itself

    Return type
        DataFlowRuntime

with_conda(conda_spec: Optional[dict] = None)

```

**with\_configuration**(*config: dict*) → *DataFlowRuntime*

Set Configuration for Spark.

**Parameters**

**config** (*dict*) – dictionary of configuration details <https://spark.apache.org/docs/latest/configuration.html#available-properties>. Example: { “spark.app.name”: “My App Name”, “spark.shuffle.io.maxRetries”: “4” }

**Returns**

runtime instance itself

**Return type**

*DataFlowRuntime*

**with\_custom\_conda**(*uri: str, region: Optional[str] = None, auth\_type: Optional[str] = None*)

Specifies the custom conda pack for running the job

**Parameters**

- **uri** (*str*) – The OCI object storage URI for the conda pack, e.g. “oci://your\_bucket@namespace/object\_name.” In the Environment Explorer of an OCI notebook session, this is shown as the “source” of the conda pack.
- **region** (*str, optional*) – The region of the bucket storing the custom conda pack, by default None. If region is not specified, ADS will use the region from your authentication credentials, \* For API Key, config[“region”] is used. \* For Resource Principal, signer.region is used. This is required if the conda pack is stored in a different region.
- **auth\_type** (*str, (=“resource\_principal”)*) – One of “resource\_principal”, “api\_keys”, “instance\_principal”, etc. Auth mechanism used to read the conda back uri provided.

**Returns**

The runtime instance.

**Return type**

self

**See also:**

**https**

[//docs.oracle.com/en-us/iaas/data-science/using/conda\\_publishs\\_object.htm](https://docs.oracle.com/en-us/iaas/data-science/using/conda_publishs_object.htm)

**with\_script\_bucket**(*bucket*) → *DataFlowRuntime*

Set object storage bucket to save the script, in case script uri given is local.

**Parameters**

**bucket** (*str*) – name of the bucket

**Returns**

runtime instance itself

**Return type**

*DataFlowRuntime*

**with\_script\_uri**(*path*) → *DataFlowRuntime*

Set script uri.

**Parameters**

**uri** (*str*) – uri to the script

**Returns**

runtime instance itself

**Return type***DataFlowRuntime***with\_service\_conda**(*slug: str*)

Specifies the service conda pack for running the job

**Parameters****slug** (*str*) – The slug name of the service conda pack**Returns**

The runtime instance.

**Return type**

self

```
class ads.jobs.builders.runtimes.python_runtime.GitPythonRuntime(spec: Optional[Dict] = None,
                                                                **kwargs)
```

Bases: *CondaRuntime*, *\_PythonRuntimeMixin*

Represents a job runtime with source code from git repository

Initialize the object with specifications.

User can either pass in the specification as a dictionary or through keyword arguments.

**Parameters**

- **spec** (*dict*, *optional*) – Object specification, by default None
- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

**CONST\_BRANCH** = 'branch'**CONST\_COMMIT** = 'commit'**CONST\_GIT\_SSH\_SECRET\_ID** = 'gitSecretId'**CONST\_GIT\_URL** = 'url'**CONST\_SKIP\_METADATA** = 'skipMetadataUpdate'

```
attribute_map = {'branch': 'branch', 'commit': 'commit', 'conda': 'conda',
'entryFunction': 'entry_function', 'entrypoint': 'entrypoint', 'env': 'env',
'freeformTags': 'freeform_tags', 'gitSecretId': 'git_secret_id', 'outputDir':
'output_dir', 'outputUri': 'output_uri', 'pythonPath': 'python_path',
'skipMetadataUpdate': 'skip_metadata_update', 'url': 'url'}
```

**property branch: str**

Git branch name.

**property commit: str**

Git commit ID (SHA1 hash)

**property skip\_metadata\_update**

Indicate if the metadata update should be skipped after the job run

By default, the job run metadata will be updated with the following freeform tags: \* repo: The URL of the Git repository \* commit: The Git commit ID \* module: The entry script/module \* method: The entry function/method \* outputs. The prefix of the output files in object storage.

This update step also requires resource principals to have the permission to update the job run.

**Returns**

True if the metadata update will be skipped. Otherwise False.

**Return type**

bool

**property ssh\_secret\_ocid**

The OCID of the OCI Vault secret storing the Git SSH key.

**property url: str**

URL of the Git repository.

**with\_argument(\*args, \*\*kwargs)**

Specifies the arguments for running the script/function.

When running a python script, the arguments will be the command line arguments. For example, `with_argument("arg1", "arg2", key1="val1", key2="val2")` will generate the command line arguments: `"arg1 arg2 -key1 val1 -key2 val2"`

When running a function, the arguments will be passed into the function. Arguments can also be list, dict or any JSON serializable object. For example, `with_argument("arg1", "arg2", key1=["val1a", "val1b"], key2="val2")` will be passed in as `"your_function("arg1", "arg2", key1=["val1a", "val1b"], key2="val2")`

**Returns**

The runtime instance.

**Return type**

self

**with\_source**(*url: str, branch: Optional[str] = None, commit: Optional[str] = None, secret\_ocid: Optional[str] = None*)

Specifies the Git repository and branch/commit for the job source code.

**Parameters**

- **url** (*str*) – URL of the Git repository.
- **branch** (*str, optional*) – Git branch name, by default None, the default branch will be used.
- **commit** (*str, optional*) – Git commit ID (SHA1 hash), by default None, the most recent commit will be used.
- **secret\_ocid** (*str*) – The secret OCID storing the SSH key content for checking out the Git repository.

**Returns**

The runtime instance.

**Return type**

self

**class** `ads.jobs.builders.runtimes.python_runtime.NotebookRuntime`(*spec: Optional[Dict] = None, \*\*kwargs*)

Bases: [`CondaRuntime`](#)

Represents a job runtime with Jupyter notebook

Initialize the object with specifications.

User can either pass in the specification as a dictionary or through keyword arguments.



**Parameters**

- **spec** (*dict*, *optional*) – Object specification, by default None
- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

**CONST\_EXCLUDE\_TAG** = 'excludeTags'

**CONST\_NOTEBOOK\_ENCODING** = 'notebookEncoding'

**CONST\_NOTEBOOK\_PATH** = 'notebookPathURI'

**CONST\_OUTPUT\_URI** = 'outputURI'

**attribute\_map** = {'conda': 'conda', 'env': 'env', 'excludeTags': 'exclude\_tags',  
'freeformTags': 'freeform\_tags', 'notebookEncoding': 'notebook\_encoding',  
'notebookPathURI': 'notebook\_path\_uri', 'outputURI': 'output\_uri'}

**property exclude\_tag: list**

A list of cell tags indicating cells to be excluded from the job

**property notebook\_encoding: str**

The encoding of the notebook

**property notebook\_uri: str**

The URI of the notebook

**property output\_uri: list**

URI for storing the output notebook and files

**with\_exclude\_tag(\*tags)**

Specifies the cell tags in the notebook to exclude cells from the job script.

**Parameters**

**\*tags** (*list*) – A list of tags (strings).

**Returns**

The runtime instance.

**Return type**

self

**with\_notebook(path: str, encoding='utf-8')**

Specifies the notebook to be converted to python script and run as a job.

**Parameters**

**path** (*str*) – The path of the Jupyter notebook

**Returns**

The runtime instance.

**Return type**

self

**with\_output(output\_uri: str)**

Specifies the output URI for storing the output notebook and files.

**Parameters**

**output\_uri** (*str*) – URI for storing the output notebook and files. For example,  
oci://bucket@namespace/path/to/dir

**Returns**

The runtime instance.

**Return type**

self

```
class ads.jobs.builders.runtimes.python_runtime.PythonRuntime(spec: Optional[Dict] = None,
                                                             **kwargs)
```

Bases: [ScriptRuntime](#), [\\_PythonRuntimeMixin](#)

Represents a job runtime using ADS driver script to run Python code

Initialize the object with specifications.

User can either pass in the specification as a dictionary or through keyword arguments.

**Parameters**

- **spec** (*dict*, *optional*) – Object specification, by default None
- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

```
CONST_WORKING_DIR = 'workingDir'
```

```
attribute_map = {'conda': 'conda', 'entryFunction': 'entry_function', 'entrypoint':  
'entrypoint', 'env': 'env', 'freeformTags': 'freeform_tags', 'outputDir':  
'output_dir', 'outputUri': 'output_uri', 'pythonPath': 'python_path',  
'scriptPathURI': 'script_path_uri', 'workingDir': 'working_dir'}
```

```
with_working_dir(working_dir: str)
```

Specifies the working directory in the job run. By default, the working directory will be the directory containing the user code (job artifact directory). This can be changed by specifying a relative path to the job artifact directory.

**Parameters**

**working\_dir** (*str*) – The path of the working directory. This can be a relative path from the job artifact directory.

**Returns**

The runtime instance.

**Return type**

self

```
property working_dir: str
```

The working directory for the job run.

```
class ads.jobs.builders.runtimes.python_runtime.ScriptRuntime(spec: Optional[Dict] = None,
                                                             **kwargs)
```

Bases: [CondaRuntime](#)

Represents job runtime with scripts and conda pack

Initialize the object with specifications.

User can either pass in the specification as a dictionary or through keyword arguments.

**Parameters**

- **spec** (*dict*, *optional*) – Object specification, by default None

- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

**CONST\_ENTRYPOINT** = 'entrypoint'

**CONST\_SCRIPT\_PATH** = 'scriptPathURI'

**attribute\_map** = {'conda': 'conda', 'entrypoint': 'entrypoint', 'env': 'env', 'freeformTags': 'freeform\_tags', 'scriptPathURI': 'script\_path\_uri'}

**property entrypoint: str**

The relative path of the script to be set as entrypoint when source is a zip/tar/directory.

**property script\_uri: str**

The URI of the source code

**property source\_uri: str**

The URI of the source code

**with\_entrypoint**(*entrypoint: str*)

Specify the entrypoint for the job

#### Parameters

**entrypoint** (*str*) – The relative path of the script to be set as entrypoint when source is a zip/tar/directory.

#### Returns

The runtime instance.

#### Return type

self

**with\_script**(*uri: str*)

Specifies the source code script for the job

#### Parameters

**uri** (*str*) – URI to the Python or Shell script, which can be any URI supported by fsspec, including <http://>, <https://> and OCI object storage. For example: `oci://your_bucket@your_namespace/path/to/script.py`

#### Returns

The runtime instance.

#### Return type

self

**with\_source**(*uri: str, entrypoint: Optional[str] = None*)

Specifies the source code for the job

#### Parameters

- **uri** (*str*) – URI to the source code, which can be a (.py/.sh) script, a zip/tar file or directory containing the scripts/modules. If the source code is a single file, URI can be any URI supported by fsspec, including <http://>, <https://> and OCI object storage. For example: `oci://your_bucket@your_namespace/path/to/script.py`. If the source code is a directory, only local directory is supported.
- **entrypoint** (*str, optional*) – The relative path of the script to be set as entrypoint when source is a zip/tar/directory. By default None. This is not needed when the source is a single script.

**Returns**

The runtime instance.

**Return type**

self

**18.1.1.13.4 ads.jobs.builders.infrastructure.dataflow module**

```
class ads.jobs.builders.infrastructure.dataflow.DataFlow(spec: Optional[dict] = None, **kwargs)
```

Bases: Infrastructure

Initialize the object with specifications.

User can either pass in the specification as a dictionary or through keyword arguments.

**Parameters**

- **spec** (*dict*, *optional*) – Object specification, by default None
- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

```
CONST_BUCKET_URI = 'logs_bucket_uri'
```

```
CONST_COMPARTMENT_ID = 'compartment_id'
```

```
CONST_CONFIG = 'configuration'
```

```
CONST_DRIVER_SHAPE = 'driver_shape'
```

```
CONST_DRIVER_SHAPE_CONFIG = 'driver_shape_config'
```

```
CONST_EXECUTE = 'execute'
```

```
CONST_EXECUTOR_SHAPE = 'executor_shape'
```

```
CONST_EXECUTOR_SHAPE_CONFIG = 'executor_shape_config'
```

```
CONST_ID = 'id'
```

```
CONST_LANGUAGE = 'language'
```

```
CONST_MEMORY_IN_GBS = 'memory_in_gbs'
```

```
CONST_METASTORE_ID = 'metastore_id'
```

```
CONST_NUM_EXECUTORS = 'num_executors'
```

```
CONST_OCPUS = 'ocpus'
```

```
CONST_SPARK_VERSION = 'spark_version'
```

```
CONST_WAREHOUSE_BUCKET_URI = 'warehouse_bucket_uri'
```

```
attribute_map = {'compartment_id': 'compartmentId', 'configuration':  
'configuration', 'driver_shape': 'driverShape', 'driver_shape_config':  
'driverShapeConfig', 'execute': 'execute', 'executor_shape': 'executorShape',  
'executor_shape_config': 'executorShapeConfig', 'id': 'id', 'logs_bucket_uri':  
'logsBucketUri', 'memory_in_gbs': 'memoryInGBs', 'metastore_id': 'metastoreId',  
'num_executors': 'numExecutors', 'ocpus': 'ocpus', 'spark_version': 'sparkVersion',  
'warehouse_bucket_uri': 'warehouseBucketUri'}
```

**create**(*runtime*: [DataFlowRuntime](#), *\*\*kwargs*) → [DataFlow](#)

Create a Data Flow job given a runtime.

**Parameters**

- **runtime** – runtime to bind to the Data Flow job
- **kwargs** – additional keyword arguments

**Returns**

a Data Flow job instance

**Return type**

[DataFlow](#)

**delete**()

Delete a Data Flow job and canceling associated runs.

**Return type**

None

**classmethod from\_dict**(*config*: *dict*) → [DataFlow](#)

Load a Data Flow job instance from a dictionary of configurations.

**Parameters**

**config** (*dict*) – dictionary of configurations

**Returns**

a Data Flow job instance

**Return type**

[DataFlow](#)

**classmethod from\_id**(*id*: *str*) → [DataFlow](#)

Load a Data Flow job given an id.

**Parameters**

**id** (*str*) – id of the Data Flow job to load

**Returns**

a Data Flow job instance

**Return type**

[DataFlow](#)

**property job\_id**: [Optional](#)[*str*]

The OCID of the job

**classmethod list\_jobs**(*compartment\_id*: [Optional](#)[*str*] = *None*, *\*\*kwargs*) → [List](#)[[DataFlow](#)]

List Data Flow jobs in a given compartment.

**Parameters**

- **compartment\_id** (*str*) – id of that compartment
- **kwargs** – additional keyword arguments for filtering jobs

**Returns**

list of Data Flow jobs

**Return type**

[List](#)[[DataFlow](#)]

**property name: str**

Display name of the job

**run**(*name: Optional[str] = None, args: Optional[List[str]] = None, env\_vars: Optional[Dict[str, str]] = None, freeform\_tags: Optional[Dict[str, str]] = None, wait: bool = False, \*\*kwargs*) → *DataFlowRun*

Run a Data Flow job.

**Parameters**

- **name** (*str, optional*) – name of the run. If a name is not provided, a randomly generated easy to remember name with timestamp will be generated, like ‘strange-spider-2022-08-17-23:55.02’.
- **args** (*List[str], optional*) – list of command line arguments
- **env\_vars** (*Dict[str, str], optional*) – dictionary of environment variables (not used for data flow)
- **freeform\_tags** (*Dict[str, str], optional*) – freeform tags
- **wait** (*bool, optional*) – whether to wait for a run to terminate
- **kwargs** – additional keyword arguments

**Returns**

a DataFlowRun instance

**Return type**

*DataFlowRun*

**run\_list**(*\*\*kwargs*) → List[*DataFlowRun*]

List runs associated with a Data Flow job.

**Parameters**

**kwargs** – additional arguments for filtering runs.

**Returns**

list of DataFlowRun instances

**Return type**

List[*DataFlowRun*]

**to\_dict**() → dict

Serialize job to a dictionary.

**Returns**

serialized job as a dictionary

**Return type**

dict

**to\_yaml**() → str

Serializes the object into YAML string.

**Returns**

YAML stored in a string.

**Return type**

str

**with\_compartment\_id**(*id: str*) → *DataFlow*

Set compartment id for a Data Flow job.

**Parameters****id** (*str*) – compartment id**Returns**

the Data Flow instance itself

**Return type***DataFlow***with\_configuration**(*configs: dict*) → *DataFlow*

Set configuration for a Data Flow job.

**Parameters****configs** (*dict*) – dictionary of configurations**Returns**

the Data Flow instance itself

**Return type***DataFlow***with\_driver\_shape**(*shape: str*) → *DataFlow*

Set driver shape for a Data Flow job.

**Parameters****shape** (*str*) – driver shape**Returns**

the Data Flow instance itself

**Return type***DataFlow***with\_driver\_shape\_config**(*memory\_in\_gbs: float, ocpus: float, \*\*kwargs: Dict[str, Any]*) → *DataFlow*Sets the driver shape config details of Data Flow job infrastructure. Specify only when a flex shape is selected. For example *VM.Standard.E3.Flex* allows the *memory\_in\_gbs* and *cpu* count to be specified.**Parameters**

- **memory\_in\_gbs** (*float*) – The size of the memory in GBs.
- **ocpus** (*float*) – The OCPUs count.
- **kwargs** – Additional keyword arguments.

**Returns**

the Data Flow instance itself.

**Return type***DataFlow***with\_execute**(*exec: str*) → *DataFlow*

Set command for spark-submit.

**Parameters****exec** (*str*) – str of commands**Returns**

the Data Flow instance itself

**Return type***DataFlow*

**with\_executor\_shape**(*shape: str*) → *DataFlow*

Set executor shape for a Data Flow job.

**Parameters**

**shape** (*str*) – executor shape

**Returns**

the Data Flow instance itself

**Return type**

*DataFlow*

**with\_executor\_shape\_config**(*memory\_in\_gbs: float, ocpus: float, \*\*kwargs: Dict[str, Any]*) → *DataFlow*

Sets the executor shape config details of Data Flow job infrastructure. Specify only when a flex shape is selected. For example *VM.Standard.E3.Flex* allows the *memory\_in\_gbs* and *cpu* count to be specified.

**Parameters**

- **memory\_in\_gbs** (*float*) – The size of the memory in GBs.
- **ocpus** (*float*) – The OCPUs count.
- **kwargs** – Additional keyword arguments.

**Returns**

the Data Flow instance itself.

**Return type**

*DataFlow*

**with\_id**(*id: str*) → *DataFlow*

Set id for a Data Flow job.

**Parameters**

**id** (*str*) – id of a job

**Returns**

the Data Flow instance itself

**Return type**

*DataFlow*

**with\_language**(*lang: str*) → *DataFlow*

Set language for a Data Flow job.

**Parameters**

**lang** (*str*) – language for the job

**Returns**

the Data Flow instance itself

**Return type**

*DataFlow*

**with\_logs\_bucket\_uri**(*uri: str*) → *DataFlow*

Set logs bucket uri for a Data Flow job.

**Parameters**

**uri** (*str*) – uri to logs bucket

**Returns**

the Data Flow instance itself



**Return type***DataFlow***with\_metastore\_id**(*id: str*) → *DataFlow*

Set Hive metastore id for a Data Flow job.

**Parameters****id** (*str*) – metastore id**Returns**

the Data Flow instance itself

**Return type***DataFlow***with\_num\_executors**(*n: int*) → *DataFlow*

Set number of executors for a Data Flow job.

**Parameters****n** (*int*) – number of executors**Returns**

the Data Flow instance itself

**Return type***DataFlow***with\_spark\_version**(*ver: str*) → *DataFlow*Set spark version for a Data Flow job. Currently supported versions are 2.4.4, 3.0.2 and 3.2.1 Documentation: [https://docs.oracle.com/en-us/iaas/data-flow/using/dfs\\_getting\\_started.htm#before\\_you\\_begin](https://docs.oracle.com/en-us/iaas/data-flow/using/dfs_getting_started.htm#before_you_begin)**Parameters****ver** (*str*) – spark version**Returns**

the Data Flow instance itself

**Return type***DataFlow***with\_warehouse\_bucket\_uri**(*uri: str*) → *DataFlow*

Set warehouse bucket uri for a Data Flow job.

**Parameters****uri** (*str*) – uri to warehouse bucket**Returns**

the Data Flow instance itself

**Return type***DataFlow*

```
class ads.jobs.builders.infrastructure.dataflow.DataFlowApp(config: Optional[dict] = None, signer: Optional[Signer] = None, client_kwargs: Optional[dict] = None, **kwargs)
```

Bases: OCIModelMixin, Application

Initializes a service/resource with OCI client as a property. If config or signer is specified, it will be used to initialize the OCI client. If neither of them is specified, the client will be initialized with `ads.common.auth.default_signer`. If both of them are specified, both of them will be passed into the OCI client,

and the authentication will be determined by OCI Python SDK.

**Parameters**

- **config** (*dict*, *optional*) – OCI API key config dictionary, by default None.
- **signer** (*oci.signer.Signer*, *optional*) – OCI authentication signer, by default None.
- **client\_kwargs** (*dict*, *optional*) – Additional keyword arguments for initializing the OCI client.

**property client:** `DataFlowClient`

OCI client

**create()** → `DataFlowApp`

Create a Data Flow application.

**Returns**

a `DataFlowApp` instance

**Return type**

`DataFlowApp`

**delete()** → None

Delete a Data Flow application.

**Return type**

None

**classmethod init\_client** (*\*\*kwargs*) → `DataFlowClient`

Initializes the OCI client specified in the “client” keyword argument Sub-class should override this method and call `cls._init_client(client=OCI_CLIENT)`

**Parameters**

**\*\*kwargs** – Additional keyword arguments for initializing the OCI client.

**Return type**

An instance of OCI client.

**to\_yaml()** → str

Serializes the object into YAML string.

**Returns**

YAML stored in a string.

**Return type**

str

**class** `ads.jobs.builders.infrastructure.dataflow.DataFlowLogs` (*run\_id*)

Bases: object

**property application**

**property driver**

**property executor**

**class** `ads.jobs.builders.infrastructure.dataflow.DataFlowRun` (*config: Optional[dict] = None*, *signer: Optional[Signer] = None*, *client\_kwargs: Optional[dict] = None*, *\*\*kwargs*)

Bases: OCIModelMixin, Run, RunInstance

Initializes a service/resource with OCI client as a property. If config or signer is specified, it will be used to initialize the OCI client. If neither of them is specified, the client will be initialized with `ads.common.auth.default_signer`. If both of them are specified, both of them will be passed into the OCI client,

and the authentication will be determined by OCI Python SDK.

#### Parameters

- **config** (*dict*, *optional*) – OCI API key config dictionary, by default None.
- **signer** (*oci.signer.Signer*, *optional*) – OCI authentication signer, by default None.
- **client\_kwargs** (*dict*, *optional*) – Additional keyword arguments for initializing the OCI client.

**TERMINATED\_STATES** = ['CANCELED', 'FAILED', 'SUCCEEDED']

**property client:** *DataFlowClient*

OCI client

**create()** → *DataFlowRun*

Create a Data Flow run.

#### Returns

a DataFlowRun instance

#### Return type

*DataFlowRun*

**delete()** → None

Cancel a Data Flow run if it is not yet terminated.

#### Return type

None

**classmethod init\_client** (*\*\*kwargs*) → *DataFlowClient*

Initializes the OCI client specified in the “client” keyword argument Sub-class should override this method and call `cls._init_client(client=OCI_CLIENT)`

#### Parameters

**\*\*kwargs** – Additional keyword arguments for initializing the OCI client.

#### Return type

An instance of OCI client.

**property logs:** *DataFlowLogs*

Show logs from a run. There are three types of logs: application log, driver log and executor log, each with stdout and stderr separately. To access each type of logs, `>>> dfr.logs.application.stdout >>> dfr.logs.driver.stderr`

#### Returns

an instance of DataFlowLogs

#### Return type

*DataFlowLogs*

**property run\_details\_link**

Link to run details page in OCI console

**Returns**

html display

**Return type**

DisplayHandle

**property status: str**

Show status (lifecycle state) of a run.

**Returns**

status of the run

**Return type**

str

**to\_yaml()** → str

Serializes the object into YAML string.

**Returns**

YAML stored in a string.

**Return type**

str

**wait**(interval: int = 3) → *DataFlowRun*

Wait for a run to terminate.

**Parameters**

**interval** (int, optional) – interval to wait before probing again

**Returns**

a DataFlowRun instance

**Return type**

*DataFlowRun*

**watch**(interval: int = 3) → *DataFlowRun*

This is an alias of *wait()* method. It waits for a run to terminate.

**Parameters**

**interval** (int, optional) – interval to wait before probing again

**Returns**

a DataFlowRun instance

**Return type**

*DataFlowRun*

### 18.1.1.13.5 ads.jobs.builders.infrastructure.dsc\_job module

```
class ads.jobs.builders.infrastructure.dsc_job.DSCJob(artifact: Optional[Union[str, Artifact]] =  
                                                    None, **kwargs)
```

Bases: OCIDataScienceMixin, Job

Represents an OCI Data Science Job This class contains all attributes of the oci.data\_science.models.Job. The main purpose of this class is to link the oci.data\_science.models.Job model and the related client methods. Mainly, linking the Job model (payload) to Create/Update/Get/List/Delete methods.

A DSCJob can be initialized by unpacking a the properties stored in a dictionary (payload):

```
job_properties = {  
    "display_name": "my_job",  
    "job_infrastructure_configuration_details": {"shape_name": "VM.MY_SHAPE"}  
}  
job = DSCJob(**job_properties)
```

The properties can also be OCI REST API payload, in which the keys are in camel format.

```
job_payload = {  
    "projectId": "<project_ocid>",  
    "compartmentId": "<compartment_ocid>",  
    "displayName": "<job_name>",  
    "jobConfigurationDetails": {  
        "jobType": "DEFAULT",  
        "commandLineArguments": "pos_arg1 pos_arg2 --key1 val1 --key2 val2",  
        "environmentVariables": {  
            "KEY1": "VALUE1",  
            "KEY2": "VALUE2",  
            # User specifies conda env via env var  
            "CONDA_ENV_TYPE" : "service",  
            "CONDA_ENV_SLUG" : "mlcpuv1"  
        }  
    },  
    "jobInfrastructureConfigurationDetails": {  
        "jobInfrastructureType": "STANDALONE",  
        "shapeName": "VM.Standard.E3.Flex",  
        "jobShapeConfigDetails": {  
            "memoryInGBs": 16,  
            "ocpus": 1  
        },  
        "blockStorageSizeInGBs": "100",  
        "subnetId": "<subnet_ocid>"  
    }  
}  
job = DSCJob(**job_payload)
```

Initialize a DSCJob object.

#### Parameters

- **artifact** (*str* or *Artifact*) – Job artifact, which can be a path or an Artifact object. Defaults to None.

- **kwargs** – Same as kwargs in `oci.data_science.models.Job`. Keyword arguments are passed into OCI Job model to initialize the properties.

**DEFAULT\_INFRA\_TYPE** = 'ME\_STANDALONE'

**property artifact:** Union[str, Artifact]

Job artifact.

**Returns**

When creating a job, this be a path or an Artifact object. When loading the job from OCI, this will be the filename of the job artifact.

**Return type**

str or Artifact

**create()** → *DSCJob*

Create the job on OCI Data Science platform

**Returns**

The DSCJob instance (self), which allows chaining additional method.

**Return type**

*DSCJob*

**delete()** → *DSCJob*

Deletes the job and the corresponding job runs.

**Returns**

The DSCJob instance (self), which allows chaining additional method.

**Return type**

*DSCJob*

**download\_artifact**(*artifact\_path*: str) → *DSCJob*

Downloads the artifact from OCI

**Parameters**

**artifact\_path** (str) – Local path to store the job artifact.

**Returns**

The DSCJob instance (self), which allows chaining additional method.

**Return type**

*DSCJob*

**classmethod from\_ocid**(*ocid*) → *DSCJob*

Gets a job by OCID

**Parameters**

**ocid** (str) – The OCID of the job.

**Returns**

An instance of DSCJob.

**Return type**

*DSCJob*

**load\_properties\_from\_env**() → None

Loads default properties from the environment

**run**(\*\*kwargs) → *DataScienceJobRun*

Runs the job

#### Parameters

- **\*\*kwargs** – Keyword arguments for initializing a Data Science Job Run. The keys can be any keys in supported by OCI JobConfigurationDetails and JobRun, including:  
 \* hyperparameter\_values: dict(str, str) \* environment\_variables: dict(str, str) \* command\_line\_arguments: str \* maximum\_runtime\_in\_minutes: int \* display\_name: str
- **specified** (If display\_name is not) –
- "<JOB\_NAME>-run-<TIMESTAMP>". (it will be generated as) –

#### Returns

An instance of DSCJobRun, which can be used to monitor the job run.

#### Return type

DSCJobRun

**run\_list**(\*\*kwargs) → list[*DataScienceJobRun*]

Lists the runs of this job.

#### Parameters

**\*\*kwargs** – Keyword arguments to be passed into the OCI list\_job\_runs() for filtering the job runs.

#### Returns

A list of DSCJobRun objects

#### Return type

list

**update**() → *DSCJob*

Updates the Data Science Job.

**upload\_artifact**(artifact\_path: Optional[str] = None) → *DSCJob*

Uploads the job artifact to OCI

#### Parameters

**artifact\_path** (str, optional) – Local path to the job artifact file to be uploaded, by default None. If artifact\_path is None, the path in self.artifact will be used.

#### Returns

The DSCJob instance (self), which allows chaining additional method.

#### Return type

*DSCJob*

ads.jobs.builders.infrastructure.dsc\_job.DSCJobRun

alias of *DataScienceJobRun*

**class** ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob(spec: Optional[Dict] = None, \*\*kwargs)

Bases: Infrastructure

Represents the OCI Data Science Job infrastructure.

Initializes a data science job infrastructure

#### Parameters

- **spec** (dict, optional) – Object specification, by default None

- **kwargs** (*dict*) – Specification as keyword arguments. If spec contains the same key as the one in kwargs, the value from kwargs will be used.

**CONST\_BLOCK\_STORAGE** = 'blockStorageSize'

**CONST\_COMPARTMENT\_ID** = 'compartmentId'

**CONST\_DISPLAY\_NAME** = 'displayName'

**CONST\_JOB\_INFRA** = 'jobInfrastructureType'

**CONST\_JOB\_TYPE** = 'jobType'

**CONST\_LOG\_GROUP\_ID** = 'logGroupId'

**CONST\_LOG\_ID** = 'logId'

**CONST\_MEMORY\_IN\_GBS** = 'memoryInGBs'

**CONST\_OCPUS** = 'ocpus'

**CONST\_PROJECT\_ID** = 'projectId'

**CONST\_SHAPE\_CONFIG\_DETAILS** = 'shapeConfigDetails'

**CONST\_SHAPE\_NAME** = 'shapeName'

**CONST\_SUBNET\_ID** = 'subnetId'

```
attribute_map = {'blockStorageSize': 'block_storage_size', 'compartmentId':  
'compartment_id', 'displayName': 'display_name', 'jobInfrastructureType':  
'job_infrastructure_type', 'jobType': 'job_type', 'logGroupId': 'log_group_id',  
'logId': 'log_id', 'projectId': 'project_id', 'shapeConfigDetails':  
'shape_config_details', 'shapeName': 'shape_name', 'subnetId': 'subnet_id'}
```

**property block\_storage\_size:** int

Block storage size for the job

**property compartment\_id:** Optional[str]

The compartment OCID

**create**(*runtime*, *\*\*kwargs*) → *DataScienceJob*

Creates a job with runtime.

**Parameters**

**runtime** (*Runtime*) – An ADS job runtime.

**Returns**

The DataScienceJob instance (self)

**Return type**

*DataScienceJob*

**delete**() → None

Deletes a job

**classmethod from\_dsc\_job**(*dsc\_job*: *DSCJob*) → *DataScienceJob*

Initialize a DataScienceJob instance from a DSCJob

**Parameters**

**dsc\_job** (*DSCJob*) – An instance of DSCJob



**Returns**

An instance of `DataScienceJob`

**Return type**

*DataScienceJob*

**classmethod** `from_id(job_id: str) → DataScienceJob`

Gets an existing job using Job OCID

**Parameters**

**job\_id** (*str*) – Job OCID

**Returns**

An instance of `DataScienceJob`

**Return type**

*DataScienceJob*

**classmethod** `instance_shapes(compartment_id: Optional[str] = None) → list`

Lists the supported shapes for running jobs in a compartment.

**Parameters**

**compartment\_id** (*str*, *optional*) – The compartment ID for running the jobs, by default `None`. This is optional in a OCI Data Science notebook session. If this is not specified, the compartment ID of the notebook session will be used.

**Returns**

A list of dictionaries containing the information of the supported shapes.

**Return type**

list

**property** `job_id: Optional[str]`

The OCID of the job

**property** `job_infrastructure_type: Optional[str]`

Job infrastructure type

**property** `job_type: Optional[str]`

Job type

**classmethod** `list_jobs(compartment_id: Optional[str] = None, **kwargs) → List[DataScienceJob]`

Lists all jobs in a compartment.

**Parameters**

- **compartment\_id** (*str*, *optional*) – The compartment ID for running the jobs, by default `None`. This is optional in a OCI Data Science notebook session. If this is not specified, the compartment ID of the notebook session will be used.
- **\*\*kwargs** – Keyword arguments to be passed into OCI `list_jobs` API for filtering the jobs.

**Returns**

A list of `DataScienceJob` object.

**Return type**

List[*DataScienceJob*]

**property** `log_group_id: str`

Log group OCID of the data science job

**Returns**

Log group OCID

**Return type**

str

**property log\_id: str**

Log OCID for the data science job.

**Returns**

Log OCID

**Return type**

str

**property name: str**

Display name of the job

```
payload_attribute_map = {'blockStorageSize':
'job_infrastructure_configuration_details.block_storage_size_in_gbs',
'compartmentId': 'compartment_id', 'displayName': 'display_name',
'jobInfrastructureType':
'job_infrastructure_configuration_details.job_infrastructure_type', 'jobType':
'job_configuration_details.job_type', 'logGroupId':
'job_log_configuration_details.log_group_id', 'logId':
'job_log_configuration_details.log_id', 'projectId': 'project_id',
'shapeConfigDetails':
'job_infrastructure_configuration_details.job_shape_config_details', 'shapeName':
'job_infrastructure_configuration_details.shape_name', 'subnetId':
'job_infrastructure_configuration_details.subnet_id'}
```

**property project\_id: Optional[str]**

Project OCID

**run**(name=None, args=None, env\_var=None, freeform\_tags=None, wait=False) → [DataScienceJobRun](#)

Runs a job on OCI Data Science job

**Parameters**

- **name** (str, optional) – The name of the job run, by default None.
- **args** (str, optional) – Command line arguments for the job run, by default None.
- **env\_var** (dict, optional) – Environment variable for the job run, by default None
- **freeform\_tags** (dict, optional) – Freeform tags for the job run, by default None
- **wait** (bool, optional) – Indicate if this method should wait for the run to finish before it returns, by default False.

**Returns**

A Data Science Job Run instance.

**Return type**

DSCJobRun

**run\_list**(\*\*kwargs) → List[[DataScienceJobRun](#)]

Gets a list of job runs.

**Parameters**

**\*\*kwargs** – Keyword arguments for filtering the job runs. These arguments will be passed to OCI API.

**Returns**

A list of job runs.

**Return type**

List[DSCJobRun]

**property shape\_config\_details:** Dict

The details for the job run shape configuration.

```
shape_config_details_attribute_map = {'memoryInGBs': 'memory_in_gbs', 'ocpus':
'ocpus'}
```

**property shape\_name:** Optional[str]

Shape name

```
snake_to_camel_map = {'block_storage_size_in_gbs': 'blockStorageSize',
'compartment_id': 'compartmentId', 'display_name': 'displayName',
'job_infrastructure_type': 'jobInfrastructureType', 'job_shape_config_details':
'shapeConfigDetails', 'job_type': 'jobType', 'log_group_id': 'logGroupId', 'log_id':
'logId', 'project_id': 'projectId', 'shape_name': 'shapeName', 'subnet_id':
'subnetId'}
```

**static standardize\_spec**(spec)

**property status:** Optional[str]

Status of the job.

**Returns**

Status of the job.

**Return type**

str

**property subnet\_id:** str

Subnet ID

**with\_block\_storage\_size**(size\_in\_gb: int) → *DataScienceJob*

Sets the block storage size in GB

**Parameters**

**size\_in\_gb** (int) – Block storage size in GB

**Returns**

The DataScienceJob instance (self)

**Return type**

*DataScienceJob*

**with\_compartment\_id**(compartment\_id: str) → *DataScienceJob*

Sets the compartment OCID

**Parameters**

**compartment\_id** (str) – The compartment OCID

**Returns**

The DataScienceJob instance (self)

**Return type**

*DataScienceJob*

**with\_job\_infrastructure\_type**(*infrastructure\_type: str*) → *DataScienceJob*

Sets the job infrastructure type

**Parameters**

**infrastructure\_type** (*str*) – Job infrastructure type as string

**Returns**

The *DataScienceJob* instance (self)

**Return type**

*DataScienceJob*

**with\_job\_type**(*job\_type: str*) → *DataScienceJob*

Sets the job type

**Parameters**

**job\_type** (*str*) – Job type as string

**Returns**

The *DataScienceJob* instance (self)

**Return type**

*DataScienceJob*

**with\_log\_group\_id**(*log\_group\_id: str*) → *DataScienceJob*

Sets the log group OCID for the data science job. If log group ID is specified but log ID is not, a new log resource will be created automatically for each job run to store the logs.

**Parameters**

**log\_group\_id** (*str*) – Log Group OCID

**Returns**

The *DataScienceJob* instance (self)

**Return type**

*DataScienceJob*

**with\_log\_id**(*log\_id: str*) → *DataScienceJob*

Sets the log OCID for the data science job. If log ID is specified, setting the log group ID (`with_log_group_id()`) is not strictly needed. ADS will look up the log group ID automatically. However, this may require additional permission, and the look up may not be available for newly created log group. Specifying both log ID (`with_log_id()`) and log group ID (`with_log_group_id()`) can avoid such lookup and speed up the job creation.

**Parameters**

**log\_id** (*str*) – Log resource OCID.

**Returns**

The *DataScienceJob* instance (self)

**Return type**

*DataScienceJob*

**with\_project\_id**(*project\_id: str*) → *DataScienceJob*

Sets the project OCID

**Parameters**

**project\_id** (*str*) – The project OCID

**Returns**

The *DataScienceJob* instance (self)

**Return type***DataScienceJob*

**with\_shape\_config\_details**(*memory\_in\_gbs*: float, *ocpus*: float, *\*\*kwargs*: Dict[str, Any]) → *DataScienceJob*

Sets the details for the job run shape configuration. Specify only when a flex shape is selected. For example *VM.Standard.E3.Flex* allows the *memory\_in\_gbs* and *cpu* count to be specified.

**Parameters**

- **memory\_in\_gbs** (float) – The size of the memory in GBs.
- **ocpus** (float) – The OCPUs count.
- **kwargs** – Additional keyword arguments.

**Returns**

The *DataScienceJob* instance (self)

**Return type***DataScienceJob*

**with\_shape\_name**(*shape\_name*: str) → *DataScienceJob*

Sets the shape name for running the job

**Parameters**

**shape\_name** (str) – Shape name

**Returns**

The *DataScienceJob* instance (self)

**Return type***DataScienceJob*

**with\_subnet\_id**(*subnet\_id*: str) → *DataScienceJob*

Sets the subnet ID

**Parameters**

**subnet\_id** (str) – Subnet ID

**Returns**

The *DataScienceJob* instance (self)

**Return type***DataScienceJob*

```
class ads.jobs.builders.infrastructure.dsc_job.DataScienceJobRun(config: Optional[dict] = None,
                                                                signer: Optional[Signer] =
                                                                None, client_kwargs:
                                                                Optional[dict] = None,
                                                                **kwargs)
```

Bases: *OCIDataScienceMixin*, *JobRun*, *RunInstance*

Represents a Data Science Job run

Initializes a service/resource with OCI client as a property. If *config* or *signer* is specified, it will be used to initialize the OCI client. If neither of them is specified, the client will be initialized with *ads.common.auth.default\_signer*. If both of them are specified, both of them will be passed into the OCI client,

and the authentication will be determined by OCI Python SDK.

**Parameters**

- **config** (*dict*, *optional*) – OCI API key config dictionary, by default None.
- **signer** (*oci.signer.Signer*, *optional*) – OCI authentication signer, by default None.
- **client\_kwargs** (*dict*, *optional*) – Additional keyword arguments for initializing the OCI client.

**TERMINAL\_STATES** = ['SUCCEEDED', 'FAILED', 'CANCELED', 'DELETED']

**cancel()** → *DataScienceJobRun*

Cancels a job run This method will wait for the job run to be canceled before returning.

**Returns**

The job run instance.

**Return type**

self

**create()** → *DataScienceJobRun*

Creates a job run

**download(to\_dir)**

Downloads files from job run output URI to local.

**Parameters**

**to\_dir** (*str*) – Local directory to which the files will be downloaded to.

**Returns**

The job run instance (self)

**Return type**

*DataScienceJobRun*

**property job**

The job instance of this run.

**Returns**

An ADS Job instance

**Return type**

*Job*

**property log\_group\_id: str**

The log group ID from OCI logging service containing the logs from the job run.

**property log\_id: str**

The log ID from OCI logging service containing the logs from the job run.

**property logging: OCILog**

The OCILog object containing the logs from the job run

**logs(limit: Optional[int] = None)** → list

Gets the logs of the job run.

**Parameters**

**limit** (*int*, *optional*) – Limit the number of logs to be returned. Defaults to None. All logs will be returned.

**Returns**

A list of log records. Each log record is a dictionary with the following keys: id, time, message.

**Return type**

list

**property status: str**

Lifecycle status

**Returns**

Status in a string.

**Return type**

str

**to\_yaml()** → str

Serializes the object into YAML string.

**Returns**

YAML stored in a string.

**Return type**

str

**watch**(interval: float = 3) → *DataScienceJobRun*

Watches the job run until it finishes. Before the job start running, this method will output the job run status. Once the job start running, the logs will be streamed until the job is success, failed or cancelled.

**Parameters**

**interval** (int) – Time interval in seconds between each request to update the logs. Defaults to 3 (seconds).

**18.1.1.13.6 Module contents****18.1.1.14 ads.model.framework other package****18.1.1.14.1 Submodules****18.1.1.14.2 ads.model.artifact module**

**exception** ads.model.artifact.**AritfactFolderStructureError**(required\_files: Tuple[str])

Bases: Exception

**exception** ads.model.artifact.**ArtifactNestedFolderError**(folder: str)

Bases: Exception

**exception** ads.model.artifact.**ArtifactRequiredFilesError**(required\_files: Tuple[str])

Bases: Exception

**class** ads.model.artifact.**ModelArtifact**(artifact\_dir: str, model\_file\_name: Optional[str] = None, reload: Optional[bool] = False)

Bases: object

The class that represents model artifacts. It is designed to help to generate and manage model artifacts.

Initializes a ModelArtifact instance.

**Parameters**

- **artifact\_dir** (str) – The local artifact folder to store the files needed for deployment.

- **model\_file\_name** ((str, optional). Defaults to *None*.) – The file name of the serialized model.
- **reload** ((bool, optional). Defaults to *False*.) – Determine whether will reload the Model into the env.

**Returns**

A ModelArtifact instance.

**Return type**

*ModelArtifact*

**Raises**

**ValueError** – If *artifact\_dir* not provided.

**classmethod from\_uri**(*uri: str, artifact\_dir: str, model\_file\_name: Optional[str] = None, force\_overwrite: Optional[bool] = False, auth: Optional[Dict] = None*)

Constructs a ModelArtifact object from the existing model artifacts.

**Parameters**

- **uri** (*str*) – The URI of source artifact folder or achive. Can be local path or OCI object storage URI.
- **artifact\_dir** (*str*) – The local artifact folder to store the files needed for deployment.
- **model\_file\_name** ((str, optional). Defaults to *None*) – The file name of the serialized model.
- **force\_overwrite** ((bool, optional). Defaults to *False*.) – Whether to overwrite existing files or not.
- **auth** ((Dict, optional). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

**Returns**

A ModelArtifact instance

**Return type**

*ModelArtifact*

**Raises**

**ValueError** – If *uri* is equal to *artifact\_dir*, and it not exists.

**prepare\_runtime\_yaml**(*inference\_conda\_env: str, inference\_python\_version: Optional[str] = None, training\_conda\_env: Optional[str] = None, training\_python\_version: Optional[str] = None, force\_overwrite: bool = False, namespace: str = 'id19sfcrra6z', bucketname: str = 'service-conda-packs'*) → None

Generate a runtime yaml file and save it to the artifact directory.

**Parameters**

- **inference\_conda\_env** ((str, optional). Defaults to *None*.) – The object storage path of conda pack which will be used in deployment. Can be either slug or object storage path of the conda pack. You can only pass in slugs if the conda pack is a service pack.
- **inference\_python\_version** ((str, optional). Defaults to *None*.) – The python version which will be used in deployment.



- **training\_conda\_env** ((*str*, optional). Defaults to None.) – The object storage path of conda pack used during training. Can be either slug or object storage path of the conda pack. You can only pass in slugs if the conda pack is a service pack.
- **training\_python\_version** ((*str*, optional). Defaults to None.) – The python version used during training.
- **force\_overwrite** ((*bool*, optional). Defaults to False.) – Whether to overwrite existing files.
- **namespace** ((*str*, optional)) – The namespace of region.
- **bucketname** ((*str*, optional)) – The bucketname of service pack.

**Raises**

**ValueError** – If neither slug or conda\_env\_uri is provided.

**Returns**

A RuntimeInfo instance.

**Return type**

*RuntimeInfo*

**prepare\_score\_py**(*jinja\_template\_filename: str*, *model\_file\_name: Optional[str] = None*, *\*\*kwargs*)

Prepares *score.py* file.

**Parameters**

- **jinja\_template\_filename** (*str*.) – The jinja template file name.
- **model\_file\_name** ((*str*, optional). Defaults to None.) – The file name of the serialized model.
- **\*\*kwargs** ((*dict*)) – use\_torch\_script: bool

**Return type**

None

**Raises**

**ValueError** – If *model\_file\_name* not provided.

**reload()**

Syncs the *score.py* to reload the model and predict function.

**Returns**

Nothing

**Return type**

None

#### 18.1.1.14.3 ads.model.generic\_model module

**class** ads.model.generic\_model.DataScienceModelType

Bases: str

**MODEL** = 'datasciencemodel'

**MODEL\_DEPLOYMENT** = 'datasciencemodeldeployment'

```
class ads.model.generic_model.FrameworkSpecificModel(estimator: Callable, artifact_dir: Optional[str]
                                                    = None, properties:
                                                    Optional[ModelProperties] = None, auth:
                                                    Optional[Dict] = None, serialize: bool = True,
                                                    **kwargs: dict)
```

Bases: [GenericModel](#)

GenericModel Constructor.

#### Parameters

- **estimator** ((Callable).) – Trained model.
- **artifact\_dir** ((str, optional). Defaults to None.) – Artifact directory to store the files needed for deployment.
- **properties** ((ModelProperties, optional). Defaults to None.) – ModelProperties object required to save and deploy model.
- **auth** ((Dict, optional). Defaults to None.) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **serialize** ((bool, optional). Defaults to True.) – Whether to serialize the model to pickle file by default. If False, you need to serialize the model manually, save it under `artifact_dir` and update the score.py manually.

**predict**(data: Optional[Any] = None, auto\_serialize\_data: bool = True, \*\*kwargs) → Dict[str, Any]

Returns prediction of input data run against the model deployment endpoint.

#### Examples

```
>>> uri = "https://github.com/pytorch/hub/raw/master/images/dog.jpg"
>>> prediction = model.predict(image=uri)['prediction']
```

```
>>> # examples on storage options
>>> prediction = model.predict(
...     image="oci://<bucket>@<tenancy>/myimage.png",
...     storage_options=ads.auth.default_signer()
... )['prediction']
```

#### Parameters

- **data** (Any) – Data for the prediction for onnx models, for local serialization method, data can be the data types that each framework support.
- **auto\_serialize\_data** (bool.) – Whether to auto serialize input data. Defaults to *False* for GenericModel, and *True* for other frameworks. *data* required to be json serializable if *auto\_serialize\_data=False*. If *auto\_serialize\_data* set to *True*, data will be serialized before sending to model deployment endpoint.
- **kwargs** – content\_type: str, used to indicate the media type of the resource. image: PIL.Image Object or uri for the image.

A valid string path for image file can be local path, http(s), oci, s3, gs.

**storage\_options: dict**

Passed to `fsspec.open` for a particular storage connection. Please see `fsspec` (<https://filesystem-spec.readthedocs.io/en/latest/api.html#fsspec.open>) for more details.

**Returns**

Dictionary with the predicted values.

**Return type**

Dict[str, Any]

**Raises**

- **NotActiveDeploymentError** – If model deployment process was not started or not finished yet.
- **ValueError** – If `data` is empty or not JSON serializable.

**verify**(*data: Optional[Any] = None, reload\_artifacts: bool = True, auto\_serialize\_data: bool = True, \*\*kwargs*) → Dict[str, Any]

Test if deployment works in local environment.

**Examples**

```
>>> uri = "https://github.com/pytorch/hub/raw/master/images/dog.jpg"
>>> prediction = model.verify(image=uri)['prediction']
```

```
>>> # examples on storage options
>>> prediction = model.verify(
...     image="oci://<bucket><tenancy>/myimage.png",
...     storage_options=ads.auth.default_signer()
... )['prediction']
```

**Parameters**

- **data** (Any) – Data used to test if deployment works in local environment.
- **reload\_artifacts** (bool. Defaults to True.) – Whether to reload artifacts or not.
- **auto\_serialize\_data** (bool.) – Whether to auto serialize input data. Defaults to *False* for *GenericModel*, and *True* for other frameworks. *data* required to be json serializable if *auto\_serialize\_data=False*. if *auto\_serialize\_data* set to *True*, data will be serialized before sending to model deployment endpoint.
- **kwargs** – *content\_type*: str, used to indicate the media type of the resource. *image*: PIL.Image Object or uri for the image.

A valid string path for image file can be local path, http(s), oci, s3, gs.

**storage\_options: dict**

Passed to `fsspec.open` for a particular storage connection. Please see `fsspec` (<https://filesystem-spec.readthedocs.io/en/latest/api.html#fsspec.open>) for more details.

**Returns**

A dictionary which contains prediction results.

**Return type**

Dict

```
class ads.model.generic_model.GenericModel(estimator: Callable, artifact_dir: Optional[str] = None,  
                                           properties: Optional[ModelProperties] = None, auth:  
                                           Optional[Dict] = None, serialize: bool = True, **kwargs:  
                                           dict)
```

Bases: MetadataMixin, Introspectable

Generic Model class which is the base class for all the frameworks including the unsupported frameworks.

**algorithm**

The algorithm of the model.

**Type**  
str

**artifact\_dir**

Artifact directory to store the files needed for deployment.

**Type**  
str

**auth**

Default authentication is set using the `ads.set_auth` API. To override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create an authentication signer to instantiate an IdentityClient object.

**Type**  
Dict

**estimator**

Any model object generated by sklearn framework

**Type**  
Callable

**framework**

The framework of the model.

**Type**  
str

**hyperparameter**

The hyperparameters of the estimator.

**Type**  
dict

**metadata\_custom**

The model custom metadata.

**Type**  
ModelCustomMetadata

**metadata\_provenance**

The model provenance metadata.

**Type**  
ModelProvenanceMetadata

**metadata\_taxonomy**

The model taxonomy metadata.

**Type**

ModelTaxonomyMetadata

**model\_artifact**

This is built by calling prepare.

**Type***ModelArtifact***model\_deployment**

A ModelDeployment instance.

**Type***ModelDeployment***model\_file\_name**

Name of the serialized model.

**Type**

str

**model\_id**

The model ID.

**Type**

str

**properties**

ModelProperties object required to save and deploy model.

**Type***ModelProperties***runtime\_info**

A RuntimeInfo instance.

**Type***RuntimeInfo***schema\_input**

Schema describes the structure of the input data.

**Type**

Schema

**schema\_output**

Schema describes the structure of the output data.

**Type**

Schema

**serialize**

Whether to serialize the model to pkl file by default. If False, you need to serialize the model manually, save it under artifact\_dir and update the score.py manually.

**Type**

bool

**version**

The framework version of the model.

**Type**

str

**delete\_deployment(...)**

Deletes the current model deployment.

**deploy(..., \\*\*kwargs)**

Deploys a model.

**from\_model\_artifact(uri, ..., \\*\*kwargs)**

Loads model from the specified folder, or zip/tar archive.

**from\_model\_catalog(model\_id, ..., \\*\*kwargs)**

Loads model from model catalog.

**from\_model\_deployment(model\_deployment\_id, ..., \\*\*kwargs)**

Loads model from model deployment.

**update\_deployment(model\_deployment\_id, ..., \\*\*kwargs)**

Updates a model deployment.

**from\_id(ocid, ..., \\*\*kwargs)**

Loads model from model OCID or model deployment OCID.

**introspect(...)**

Runs model introspection.

**predict(data, ...)**

Returns prediction of input data run against the model deployment endpoint.

**prepare(..., \\*\*kwargs)**

Prepare and save the score.py, serialized model and runtime.yaml file.

**prepare\_save\_deploy(..., \\*\*kwargs)**

Shortcut for prepare, save and deploy steps.

**reload(...)**Reloads the model artifact files: *score.py* and the *runtime.yaml*.**save(..., \\*\*kwargs)**

Saves model artifacts to the model catalog.

**summary\_status(...)**

Gets a summary table of the current status.

**verify(data, ...)**

Tests if deployment works in local environment.

**upload\_artifact(...)**Uploads model artifacts to the provided *uri*.

## Examples

```
>>> import tempfile
>>> from ads.model.generic_model import GenericModel
```

```
>>> class Toy:
...     def predict(self, x):
...         return x ** 2
>>> estimator = Toy()
```

```
>>> model = GenericModel(estimator=estimator, artifact_dir=tempfile.mkdtemp())
>>> model.summary_status()
>>> model.prepare(
...     inference_conda_env="dataexpl_p37_cpu_v3",
...     inference_python_version="3.7",
...     model_file_name="toy_model.pkl",
...     training_id=None,
...     force_overwrite=True
... )
>>> model.verify(2)
>>> model.save()
>>> model.deploy()
>>> # Update access log id, freeform tags and description for the model deployment
>>> model.update_deployment(
>>>     properties=ModelDeploymentProperties(
>>>         access_log_id=<log_ocid>,
>>>         description="Description for Custom Model",
>>>         freeform_tags={"key": "value"},
>>>     )
>>> )
>>> model.predict(2)
>>> # Uncomment the line below to delete the model and the associated model_
↪ deployment
>>> # model.delete(delete_associated_model_deployment = True)
```

GenericModel Constructor.

### Parameters

- **estimator** ((Callable).) – Trained model.
- **artifact\_dir** ((str, optional). Defaults to None.) – Artifact directory to store the files needed for deployment.
- **properties** ((ModelProperties, optional). Defaults to None.) – ModelProperties object required to save and deploy model.
- **auth** ((Dict, optional). Defaults to None.) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **serialize** ((bool, optional). Defaults to True.) – Whether to serialize the model to pkl file by default. If False, you need to serialize the model manually, save it under `artifact_dir` and update the score.py manually.

```
classmethod delete(model_id: Optional[str] = None, delete_associated_model_deployment:  
Optional[bool] = False, delete_model_artifact: Optional[bool] = False, artifact_dir:  
Optional[str] = None, **kwargs: Dict) → None
```

Deletes a model from Model Catalog.

#### Parameters

- **model\_id** ((*str*, optional). Defaults to *None*.) – The model OCID to be deleted. If the method called on instance level, then *self.model\_id* will be used.
- **delete\_associated\_model\_deployment** ((*bool*, optional). Defaults to *False*.) – Whether associated model deployments need to be deleted or not.
- **delete\_model\_artifact** ((*bool*, optional). Defaults to *False*.) – Whether associated model artifacts need to be deleted or not.
- **artifact\_dir** ((*str*, optional). Defaults to *None*) – The local path to the model artifacts folder. If the method called on instance level, the *self.artifact\_dir* will be used by default.

#### Return type

None

#### Raises

**ValueError** – If *model\_id* not provided.

```
delete_deployment(wait_for_completion: bool = True) → None
```

Deletes the current deployment.

#### Parameters

**wait\_for\_completion** ((*bool*, optional). Defaults to *True*.) – Whether to wait till completion.

#### Return type

None

#### Raises

**ValueError** – if there is not deployment attached yet.:

```
deploy(wait_for_completion: Optional[bool] = True, display_name: Optional[str] = None, description:  
Optional[str] = None, deployment_instance_shape: Optional[str] = None,  
deployment_instance_count: Optional[int] = None, deployment_bandwidth_mbps: Optional[int] =  
None, deployment_log_group_id: Optional[str] = None, deployment_access_log_id: Optional[str] =  
None, deployment_predict_log_id: Optional[str] = None, deployment_memory_in_gbs:  
Optional[float] = None, deployment_ocpus: Optional[float] = None, **kwargs: Dict) →  
ModelDeployment
```

Deploys a model. The model needs to be saved to the model catalog at first.

#### Parameters

- **wait\_for\_completion** ((*bool*, optional). Defaults to *True*.) – Flag set for whether to wait for deployment to complete before proceeding.
- **display\_name** ((*str*, optional). Defaults to *None*.) – The name of the model. If a *display\_name* is not provided in kwargs, a randomly generated easy to remember name with timestamp will be generated, like 'strange-spider-2022-08-17-23:55.02'.
- **description** ((*str*, optional). Defaults to *None*.) – The description of the model.
- **deployment\_instance\_shape** ((*str*, optional). Default to *VM.Standard2.1*.) – The shape of the instance used for deployment.



- **deployment\_instance\_count** ((*int*, *optional*). Defaults to 1.) – The number of instance used for deployment.
- **deployment\_bandwidth\_mbps** ((*int*, *optional*). Defaults to 10.) – The bandwidth limit on the load balancer in Mbps.
- **deployment\_memory\_in\_gbs** ((*float*, *optional*). Defaults to None.) – Specifies the size of the memory of the model deployment instance in GBs.
- **deployment\_ocpus** ((*float*, *optional*). Defaults to None.) – Specifies the ocpu count of the model deployment instance.
- **deployment\_log\_group\_id** ((*str*, *optional*). Defaults to None.) – The oci logging group id. The access log and predict log share the same log group.
- **deployment\_access\_log\_id** ((*str*, *optional*). Defaults to None.) – The access log OCID for the access logs. [https://docs.oracle.com/en-us/iaas/data-science/using/model\\_dep\\_using\\_logging.htm](https://docs.oracle.com/en-us/iaas/data-science/using/model_dep_using_logging.htm)
- **deployment\_predict\_log\_id** ((*str*, *optional*). Defaults to None.) – The predict log OCID for the predict logs. [https://docs.oracle.com/en-us/iaas/data-science/using/model\\_dep\\_using\\_logging.htm](https://docs.oracle.com/en-us/iaas/data-science/using/model_dep_using_logging.htm)
- **kwargs** –

**project\_id:** (*str*, *optional*).

Project OCID. If not specified, the value will be taken from the environment variables.

**compartment\_id**

[(*str*, *optional*).] Compartment OCID. If not specified, the value will be taken from the environment variables.

**max\_wait\_time**

[(*int*, *optional*). Defaults to 1200 seconds.] Maximum amount of time to wait in seconds. Negative implies infinite wait time.

**poll\_interval**

[(*int*, *optional*). Defaults to 60 seconds.] Poll interval in seconds.

**freeform\_tags:** (*Dict*[*str*, *str*], *optional*). Defaults to None.

Freeform tags of the model deployment.

**defined\_tags:** (*Dict*[*str*, *dict*[*str*, *object*]], *optional*). Defaults to None.

Defined tags of the model deployment.

Also can be any keyword argument for initializing the *ads.model.deployment.ModelDeploymentProperties*. See

*ads.model.deployment.ModelDeploymentProperties()* for details.

#### Returns

The *ModelDeployment* instance.

#### Return type

*ModelDeployment*

#### Raises

**ValueError** – If *model\_id* is not specified.

```
classmethod from_id(ocid: str, model_file_name: Optional[str] = None, artifact_dir: Optional[str] =
                    None, auth: Optional[Dict] = None, force_overwrite: Optional[bool] = False,
                    properties: Optional[Union[ModelProperties, Dict]] = None, bucket_uri:
                    Optional[str] = None, remove_existing_artifact: Optional[bool] = True, **kwargs)
                    → GenericModel
```

Loads model from model OCID or model deployment OCID.

#### Parameters

- **ocid** (*str*) – The model OCID or model deployment OCID.
- **model\_file\_name** ((*str*, optional). Defaults to *None*.) – The name of the serialized model.
- **artifact\_dir** ((*str*, optional). Defaults to *None*.) – The artifact directory to store the files needed for deployment. Will be created if not exists.
- **auth** ((*Dict*, optional). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **force\_overwrite** ((*bool*, optional). Defaults to *False*.) – Whether to overwrite existing files or not.
- **properties** ((*ModelProperties*, optional). Defaults to *None*.) – Model-Properties object required to save and deploy model.
- **bucket\_uri** ((*str*, optional). Defaults to *None*.) – The OCI Object Storage URI where model artifacts will be copied to. The *bucket\_uri* is only necessary for downloading large artifacts with size is greater than 2GB. Example: *oci://<bucket\_name>@<namespace>/prefix/*.
- **remove\_existing\_artifact** ((*bool*, optional). Defaults to *True*.) – Whether artifacts uploaded to object storage bucket need to be removed or not.
- **kwargs** –
  - compartment\_id**  
[(*str*, optional)] Compartment OCID. If not specified, the value will be taken from the environment variables.
  - timeout**  
[(*int*, optional). Defaults to 10 seconds.] The connection timeout in seconds for the client.

#### Returns

An instance of GenericModel class.

#### Return type

*GenericModel*

```
classmethod from_model_artifact(uri: str, model_file_name: Optional[str] = None, artifact_dir:
                                Optional[str] = None, auth: Optional[Dict] = None,
                                force_overwrite: Optional[bool] = False, properties:
                                Optional[ModelProperties] = None, **kwargs: dict) →
                                GenericModel
```

Loads model from a folder, or zip/tar archive.

#### Parameters

- **uri** (*str*) – The folder path, ZIP file path, or TAR file path. It could contain a serialized model(required) as well as any files needed for deployment including: serialized model, runtime.yaml, score.py and etc. The content of the folder will be copied to the *artifact\_dir* folder.
- **model\_file\_name** ((*str*, optional). Defaults to *None*.) – The serialized model file name. Will be extracted from artifacts if not provided.

- **artifact\_dir** ((str, optional). Defaults to *None*.) – The artifact directory to store the files needed for deployment. Will be created if not exists.
- **auth** ((Dict, optional). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **force\_overwrite** ((bool, optional). Defaults to *False*.) – Whether to overwrite existing files or not.
- **properties** ((ModelProperties, optional). Defaults to *None*.) – Model-Properties object required to save and deploy model.

**Returns**

An instance of *GenericModel* class.

**Return type**

*GenericModel*

**Raises**

**ValueError** – If *model\_file\_name* not provided.

```
classmethod from_model_catalog(model_id: str, model_file_name: Optional[str] = None, artifact_dir:
    Optional[str] = None, auth: Optional[Dict] = None, force_overwrite:
    Optional[bool] = False, properties:
    Optional[Union[ModelProperties, Dict]] = None, bucket_uri:
    Optional[str] = None, remove_existing_artifact: Optional[bool] =
    True, **kwargs) → GenericModel
```

Loads model from model catalog.

**Parameters**

- **model\_id** (str) – The model OCID.
- **model\_file\_name** ((str, optional). Defaults to *None*.) – The name of the serialized model.
- **artifact\_dir** ((str, optional). Defaults to *None*.) – The artifact directory to store the files needed for deployment. Will be created if not exists.
- **auth** ((Dict, optional). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **force\_overwrite** ((bool, optional). Defaults to *False*.) – Whether to overwrite existing files or not.
- **properties** ((ModelProperties, optional). Defaults to *None*.) – Model-Properties object required to save and deploy model.
- **bucket\_uri** ((str, optional). Defaults to *None*.) – The OCI Object Storage URI where model artifacts will be copied to. The *bucket\_uri* is only necessary for downloading large artifacts with size is greater than 2GB. Example: *oci://<bucket\_name>@<namespace>/prefix/*.
- **remove\_existing\_artifact** ((bool, optional). Defaults to *True*.) – Whether artifacts uploaded to object storage bucket need to be removed or not.
- **kwargs** –

**compartment\_id**

[(str, optional)] Compartment OCID. If not specified, the value will be taken from the environment variables.

**timeout**

[(int, optional). Defaults to 10 seconds.] The connection timeout in seconds for the client.

**region: (str, optional). Defaults to *None*.**

The destination Object Storage bucket region. By default the value will be extracted from the *OCI\_REGION\_METADATA* environment variables.

**Returns**

An instance of GenericModel class.

**Return type**

*GenericModel*

```
classmethod from_model_deployment(model_deployment_id: str, model_file_name: Optional[str] =
                                   None, artifact_dir: Optional[str] = None, auth: Optional[Dict] =
                                   None, force_overwrite: Optional[bool] = False, properties:
                                   Optional[Union[ModelProperties, Dict]] = None, bucket_uri:
                                   Optional[str] = None, remove_existing_artifact: Optional[bool] =
                                   True, **kwargs) → GenericModel
```

Loads model from model deployment.

**Parameters**

- **model\_deployment\_id** (str) – The model deployment OCID.
- **model\_file\_name** ((str, optional). Defaults to *None*.) – The name of the serialized model.
- **artifact\_dir** ((str, optional). Defaults to *None*.) – The artifact directory to store the files needed for deployment. Will be created if not exists.
- **auth** ((Dict, optional). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.
- **force\_overwrite** ((bool, optional). Defaults to *False*.) – Whether to overwrite existing files or not.
- **properties** ((ModelProperties, optional). Defaults to *None*.) – ModelProperties object required to save and deploy model.
- **bucket\_uri** ((str, optional). Defaults to *None*.) – The OCI Object Storage URI where model artifacts will be copied to. The *bucket\_uri* is only necessary for downloading large artifacts with size is greater than 2GB. Example: *oci://<bucket\_name>@<namespace>/prefix/*.
- **remove\_existing\_artifact** ((bool, optional). Defaults to *True*.) – Whether artifacts uploaded to object storage bucket need to be removed or not.
- **kwargs** –

**compartment\_id**

[(str, optional)] Compartment OCID. If not specified, the value will be taken from the environment variables.

**timeout**

[(int, optional). Defaults to 10 seconds.] The connection timeout in seconds for the client.

**region: (str, optional). Defaults to *None*.**

The destination Object Storage bucket region. By default the value will be extracted from the *OCI\_REGION\_METADATA* environment variables.

**Returns**

An instance of GenericModel class.

**Return type**

*GenericModel*

**get\_data\_serializer**(*data: any, data\_type: Optional[str] = None*)

The data\_serializer\_class class is set in init and used here. Frameworks should subclass the InputDataSerializer class, then set that as the self.data\_serializer\_class. Frameworks should avoid overwriting this method whenever possible.

**Parameters**

- **data** ((Any)) – data to be passed to model for prediction.
- **data\_type** (str) – Type of the data.

**Returns**

Serialized data.

**Return type**

data

**introspect**() → DataFrame

Conducts introspection.

**Returns**

A pandas DataFrame which contains the introspection results.

**Return type**

pandas.DataFrame

**property metadata\_custom**

**property metadata\_provenance**

**property metadata\_taxonomy**

**property model\_deployment\_id**

**property model\_id**

**predict**(*data: Optional[Any] = None, auto\_serialize\_data: bool = False, \*\*kwargs*) → Dict[str, Any]

Returns prediction of input data run against the model deployment endpoint.

## Examples

```
>>> uri = "https://github.com/pytorch/hub/raw/master/images/dog.jpg"
>>> prediction = model.predict(image=uri)['prediction']
```

```
>>> # examples on storage options
>>> prediction = model.predict(
...     image="oci://<bucket>@<tenancy>/myimage.png",
...     storage_options=ads.auth.default_signer()
... )['prediction']
```

## Parameters

- **data** (*Any*) – Data for the prediction for onnx models, for local serialization method, data can be the data types that each framework support.
- **auto\_serialize\_data** (*bool*.) – Whether to auto serialize input data. Defaults to *False* for *GenericModel*, and *True* for other frameworks. *data* required to be json serializable if *auto\_serialize\_data=False*. If *auto\_serialize\_data* set to *True*, data will be serialized before sending to model deployment endpoint.
- **kwargs** – *content\_type*: *str*, used to indicate the media type of the resource. *image*: *PIL.Image* Object or *uri* for the image.

A valid string path for image file can be local path, http(s), oci, s3, gs.

### storage\_options: dict

Passed to *fsspec.open* for a particular storage connection. Please see *fsspec* (<https://filesystem-spec.readthedocs.io/en/latest/api.html#fsspec.open>) for more details.

## Returns

Dictionary with the predicted values.

## Return type

Dict[str, Any]

## Raises

- **NotActiveDeploymentError** – If model deployment process was not started or not finished yet.
- **ValueError** – If *data* is empty or not JSON serializable.

**prepare**(*inference\_conda\_env*: Optional[str] = None, *inference\_python\_version*: Optional[str] = None, *training\_conda\_env*: Optional[str] = None, *training\_python\_version*: Optional[str] = None, *model\_file\_name*: Optional[str] = None, *as\_onnx*: bool = False, *initial\_types*: Optional[List[Tuple]] = None, *force\_overwrite*: bool = False, *namespace*: str = 'id19sferra6z', *use\_case\_type*: Optional[str] = None, *X\_sample*: Optional[Union[list, tuple, DataFrame, Series, ndarray]] = None, *y\_sample*: Optional[Union[list, tuple, DataFrame, Series, ndarray]] = None, *training\_script\_path*: Optional[str] = None, *training\_id*: Optional[str] = None, *ignore\_pending\_changes*: bool = True, *max\_col\_num*: int = 2000, *\*\*kwargs*: Dict) → *GenericModel*

Prepare and save the score.py, serialized model and runtime.yaml file.

## Parameters

- **inference\_conda\_env** ((*str*, optional). Defaults to *None*.) – Can be either slug or object storage path of the conda pack. You can only pass in slugs if the conda pack is a service pack.
- **inference\_python\_version** ((*str*, optional). Defaults to *None*.) – Python version which will be used in deployment.
- **training\_conda\_env** ((*str*, optional). Defaults to *None*.) – Can be either slug or object storage path of the conda pack. You can only pass in slugs if the conda pack is a service pack. If *training\_conda\_env* is not provided, *training\_conda\_env* will use the same value of *training\_conda\_env*.
- **training\_python\_version** ((*str*, optional). Defaults to *None*.) – Python version used during training.
- **model\_file\_name** ((*str*, optional). Defaults to *None*.) – Name of the serialized model. Will be auto generated if not provided.
- **as\_onnx** ((*bool*, optional). Defaults to *False*.) – Whether to serialize as onnx model.
- **initial\_types** ((*list*[*Tuple*], optional).) – Defaults to *None*. Only used for *SklearnModel*, *LightGBMModel* and *XGBoostModel*. Each element is a tuple of a variable name and a type. Check this link [http://onnx.ai/sklearn-onnx/api\\_summary.html#id2](http://onnx.ai/sklearn-onnx/api_summary.html#id2) for more explanation and examples for *initial\_types*.
- **force\_overwrite** ((*bool*, optional). Defaults to *False*.) – Whether to overwrite existing files.
- **namespace** ((*str*, optional).) – Namespace of region. This is used for identifying which region the service pack is from when you pass a slug to *inference\_conda\_env* and *training\_conda\_env*.
- **use\_case\_type** (*str*) – The use case type of the model. Use it through *UseCaseType* class or string provided in *UseCaseType*. For example, `use_case_type=UseCaseType.BINARY_CLASSIFICATION` or `use_case_type="binary_classification"`. Check with *UseCaseType* class to see all supported types.
- **X\_sample** (*Union*[*list*, *tuple*, *pd.Series*, *np.ndarray*, *pd.DataFrame*]. Defaults to *None*.) – A sample of input data that will be used to generate input schema.
- **y\_sample** (*Union*[*list*, *tuple*, *pd.Series*, *np.ndarray*, *pd.DataFrame*]. Defaults to *None*.) – A sample of output data that will be used to generate output schema.
- **training\_script\_path** (*str*. Defaults to *None*.) – Training script path.
- **training\_id** ((*str*, optional). Defaults to value from environment variables.) – The training OCID for model. Can be notebook session or job OCID.
- **ignore\_pending\_changes** (*bool*. Defaults to *False*.) – whether to ignore the pending changes in the git.
- **max\_col\_num** ((*int*, optional). Defaults to *utils.DATA\_SCHEMA\_MAX\_COL\_NUM*.) – Do not generate the input schema if the input has more than this number of features(columns).
- **kwargs** –

**impute\_values: (dict, optional).**

The dictionary where the key is the column index(or names is accepted for pandas dataframe) and the value is the impute value for the corresponding column.

**Raises**

- **FileExistsError** – If files already exist but *force\_overwrite* is False.
- **ValueError** – If *inference\_python\_version* is not provided, but also cannot be found through manifest file.

**Returns**

An instance of *GenericModel* class.

**Return type**

*GenericModel*

**prepare\_save\_deploy**(*inference\_conda\_env: Optional[str] = None, inference\_python\_version: Optional[str] = None, training\_conda\_env: Optional[str] = None, training\_python\_version: Optional[str] = None, model\_file\_name: Optional[str] = None, as\_onnx: bool = False, initial\_types: Optional[List[Tuple]] = None, force\_overwrite: bool = False, namespace: str = 'id19sfcrra6z', use\_case\_type: Optional[str] = None, X\_sample: Optional[Union[list, tuple, DataFrame, Series, ndarray]] = None, y\_sample: Optional[Union[list, tuple, DataFrame, Series, ndarray]] = None, training\_script\_path: Optional[str] = None, training\_id: Optional[str] = None, ignore\_pending\_changes: bool = True, max\_col\_num: int = 2000, model\_display\_name: Optional[str] = None, model\_description: Optional[str] = None, model\_freeform\_tags: Optional[dict] = None, model\_defined\_tags: Optional[dict] = None, ignore\_introspection: Optional[bool] = False, wait\_for\_completion: Optional[bool] = True, deployment\_display\_name: Optional[str] = None, deployment\_description: Optional[str] = None, deployment\_instance\_shape: Optional[str] = None, deployment\_instance\_count: Optional[int] = None, deployment\_bandwidth\_mbps: Optional[int] = None, deployment\_log\_group\_id: Optional[str] = None, deployment\_access\_log\_id: Optional[str] = None, deployment\_predict\_log\_id: Optional[str] = None, deployment\_memory\_in\_gbs: Optional[float] = None, deployment\_ocpus: Optional[float] = None, bucket\_uri: Optional[str] = None, overwrite\_existing\_artifact: Optional[bool] = True, remove\_existing\_artifact: Optional[bool] = True, \*\*kwargs: Dict*) → *ModelDeployment*

Shortcut for prepare, save and deploy steps.

**Parameters**

- **inference\_conda\_env** ((*str*, optional). Defaults to *None*.) – Can be either slug or object storage path of the conda pack. You can only pass in slugs if the conda pack is a service pack.
- **inference\_python\_version** ((*str*, optional). Defaults to *None*.) – Python version which will be used in deployment.
- **training\_conda\_env** ((*str*, optional). Defaults to *None*.) – Can be either slug or object storage path of the conda pack. You can only pass in slugs if the conda pack is a service pack. If *training\_conda\_env* is not provided, *training\_conda\_env* will use the same value of *training\_conda\_env*.
- **training\_python\_version** ((*str*, optional). Defaults to *None*.) – Python version used during training.
- **model\_file\_name** ((*str*, optional). Defaults to *None*.) – Name of the serialized model.



- **as\_onnx** *((bool, optional). Defaults to False.)* – Whether to serialize as onnx model.
- **initial\_types** *((list[Tuple], optional).)* – Defaults to None. Only used for SklearnModel, LightGBMModel and XGBoostModel. Each element is a tuple of a variable name and a type. Check this link [http://onnx.ai/sklearn-onnx/api\\_summary.html#id2](http://onnx.ai/sklearn-onnx/api_summary.html#id2) for more explanation and examples for *initial\_types*.
- **force\_overwrite** *((bool, optional). Defaults to False.)* – Whether to overwrite existing files.
- **namespace** *((str, optional).)* – Namespace of region. This is used for identifying which region the service pack is from when you pass a slug to *inference\_conda\_env* and *training\_conda\_env*.
- **use\_case\_type** *(str)* – The use case type of the model. Use it through *UseCaseType* class or string provided in *UseCaseType*. For example, `use_case_type=UseCaseType.BINARY_CLASSIFICATION` or `use_case_type="binary_classification"`. Check with *UseCaseType* class to see all supported types.
- **X\_sample** *(Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]. Defaults to None.)* – A sample of input data that will be used to generate input schema.
- **y\_sample** *(Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]. Defaults to None.)* – A sample of output data that will be used to generate output schema.
- **training\_script\_path** *(str. Defaults to None.)* – Training script path.
- **training\_id** *((str, optional). Defaults to value from environment variables.)* – The training OCID for model. Can be notebook session or job OCID.
- **ignore\_pending\_changes** *(bool. Defaults to False.)* – whether to ignore the pending changes in the git.
- **max\_col\_num** *((int, optional). Defaults to utils.DATA\_SCHEMA\_MAX\_COL\_NUM.)* – Do not generate the input schema if the input has more than this number of features(columns).
- **model\_display\_name** *((str, optional). Defaults to None.)* – The name of the model. If a *model\_display\_name* is not provided in *kwargs*, a randomly generated easy to remember name with timestamp will be generated, like 'strange-spider-2022-08-17-23:55.02'.
- **model\_description** *((str, optional). Defaults to None.)* – The description of the model.
- **model\_freeform\_tags** *(Dict(str, str), Defaults to None.)* – Freeform tags for the model.
- **model\_defined\_tags** *((Dict(str, dict(str, object)), optional). Defaults to None.)* – Defined tags for the model.
- **ignore\_introspection** *((bool, optional). Defaults to None.)* – Determine whether to ignore the result of model introspection or not. If set to True, the save will ignore all model introspection errors.
- **wait\_for\_completion** *((bool, optional). Defaults to True.)* – Flag set for whether to wait for deployment to complete before proceeding.

- **deployment\_display\_name** ((*str*, optional). Defaults to *None*.) – The name of the model deployment. If a `deployment_display_name` is not provided in `kwargs`, a randomly generated easy to remember name with timestamp will be generated, like ‘strange-spider-2022-08-17-23:55.02’.
- **description** ((*str*, optional). Defaults to *None*.) – The description of the model.
- **deployment\_instance\_shape** ((*str*, optional). Default to *VM.Standard2.1*.) – The shape of the instance used for deployment.
- **deployment\_instance\_count** ((*int*, optional). Defaults to *1*.) – The number of instance used for deployment.
- **deployment\_bandwidth\_mbps** ((*int*, optional). Defaults to *10*.) – The bandwidth limit on the load balancer in Mbps.
- **deployment\_log\_group\_id** ((*str*, optional). Defaults to *None*.) – The oci logging group id. The access log and predict log share the same log group.
- **deployment\_access\_log\_id** ((*str*, optional). Defaults to *None*.) – The access log OCID for the access logs. [https://docs.oracle.com/en-us/iaas/data-science/using/model\\_dep\\_using\\_logging.htm](https://docs.oracle.com/en-us/iaas/data-science/using/model_dep_using_logging.htm)
- **deployment\_predict\_log\_id** ((*str*, optional). Defaults to *None*.) – The predict log OCID for the predict logs. [https://docs.oracle.com/en-us/iaas/data-science/using/model\\_dep\\_using\\_logging.htm](https://docs.oracle.com/en-us/iaas/data-science/using/model_dep_using_logging.htm)
- **deployment\_memory\_in\_gbs** ((*float*, optional). Defaults to *None*.) – Specifies the size of the memory of the model deployment instance in GBs.
- **deployment\_ocpus** ((*float*, optional). Defaults to *None*.) – Specifies the ocpu count of the model deployment instance.
- **bucket\_uri** ((*str*, optional). Defaults to *None*.) – The OCI Object Storage URI where model artifacts will be copied to. The *bucket\_uri* is only necessary for downloading large artifacts with size is greater than 2GB. Example: *oci://<bucket\_name>@<namespace>/prefix/*.
- **overwrite\_existing\_artifact** ((*bool*, optional). Defaults to *True*.) – Overwrite target bucket artifact if exists.
- **remove\_existing\_artifact** ((*bool*, optional). Defaults to *True*.) – Whether artifacts uploaded to object storage bucket need to be removed or not.

- **kwargs** –

- **impute\_values**: (*dict*, optional).

- The dictionary where the key is the column index(or names is accepted for pandas dataframe) and the value is the impute value for the corresponding column.

- **project\_id**: (*str*, optional).

- Project OCID. If not specified, the value will be taken either from the environment variables or model properties.

- **compartment\_id**

- [(*str*, optional).] Compartment OCID. If not specified, the value will be taken either from the environment variables or model properties.

- **timeout**: (*int*, optional). Defaults to **10 seconds**.

- The connection timeout in seconds for the client.

**max\_wait\_time**

[(int, optional). Defaults to 1200 seconds.] Maximum amount of time to wait in seconds. Negative implies infinite wait time.

**poll\_interval**

[(int, optional). Defaults to 60 seconds.] Poll interval in seconds.

**freeform\_tags: (Dict[str, str], optional). Defaults to None.**

Freeform tags of the model deployment.

**defined\_tags: (Dict[str, dict[str, object]], optional). Defaults to None.**

Defined tags of the model deployment.

**region: (str, optional). Defaults to None.**

The destination Object Storage bucket region. By default the value will be extracted from the `OCI_REGION_METADATA` environment variables.

Also can be any keyword argument for initializing the `ads.model.deployment.ModelDeploymentProperties`. See `ads.model.deployment.ModelDeploymentProperties()` for details.

**Returns**

The ModelDeployment instance.

**Return type**

*ModelDeployment*

**Raises**

- **FileExistsError** – If files already exist but `force_overwrite` is False.
- **ValueError** – If `inference_python_version` is not provided, but also cannot be found through manifest file.

**reload()** → *GenericModel*

Reloads the model artifact files: `score.py` and the `runtime.yaml`.

**Returns**

An instance of GenericModel class.

**Return type**

*GenericModel*

**reload\_runtime\_info()** → None

Reloads the model artifact file: `runtime.yaml`.

**Returns**

Nothing.

**Return type**

None

**save**(*display\_name: Optional[str] = None, description: Optional[str] = None, freeform\_tags: Optional[dict] = None, defined\_tags: Optional[dict] = None, ignore\_introspection: Optional[bool] = False, bucket\_uri: Optional[str] = None, overwrite\_existing\_artifact: Optional[bool] = True, remove\_existing\_artifact: Optional[bool] = True, \*\*kwargs*) → str

Saves model artifacts to the model catalog.

**Parameters**

- **display\_name** ((str, optional). Defaults to None.) – The name of the model. If a display\_name is not provided in kwargs, randomly generated easy to remember name with timestamp will be generated, like ‘strange-spider-2022-08-17-23:55.02’.

- **description** ((*str*, *optional*). Defaults to *None*.) – The description of the model.
- **freeform\_tags** (*Dict(str, str)*, Defaults to *None*.) – Freeform tags for the model.
- **defined\_tags** ((*Dict(str, dict(str, object))*, *optional*). Defaults to *None*.) – Defined tags for the model.
- **ignore\_introspection** ((*bool*, *optional*). Defaults to *None*.) – Determine whether to ignore the result of model introspection or not. If set to *True*, the save will ignore all model introspection errors.
- **bucket\_uri** ((*str*, *optional*). Defaults to *None*.) – The OCI Object Storage URI where model artifacts will be copied to. The *bucket\_uri* is only necessary for uploading large artifacts which size is greater than 2GB. Example: *oci://<bucket\_name>@<namespace>/prefix/*.
- **overwrite\_existing\_artifact** ((*bool*, *optional*). Defaults to *True*.) – Overwrite target bucket artifact if exists.
- **remove\_existing\_artifact** ((*bool*, *optional*). Defaults to *True*.) – Whether artifacts uploaded to object storage bucket need to be removed or not.
- **kwargs** –

**project\_id:** (*str*, *optional*).

Project OCID. If not specified, the value will be taken either from the environment variables or model properties.

**compartment\_id**

[(*str*, *optional*).] Compartment OCID. If not specified, the value will be taken either from the environment variables or model properties.

**region:** (*str*, *optional*). Defaults to *None*.

The destination Object Storage bucket region. By default the value will be extracted from the *OCI\_REGION\_METADATA* environment variables.

Also can be any attribute that *oci.data\_science.models.Model* accepts.

#### Raises

***RuntimeInfoInconsistencyError*** – When *.runtime\_info* is not synched with *runtime.yaml* file.

#### Returns

The model id.

#### Return type

*str*

**property schema\_input**

**property schema\_output**

**serialize\_model** (*as\_onnx: bool = False*, *initial\_types: Optional[List[Tuple]] = None*, *force\_overwrite: bool = False*, *X\_sample: Optional[any] = None*)

Serialize and save model using ONNX or model specific method.

#### Parameters

- **as\_onnx** ((*boolean*, *optional*)) – If set as *True*, convert into ONNX model.

- **initial\_types** ((*List[Tuple]*, *optional*)) – a python list. Each element is a tuple of a variable name and a data type.
- **force\_overwrite** ((*boolean*, *optional*)) – If set as True, overwrite serialized model if exists.
- **X\_sample** ((*any*, *optional*). *Defaults to None.*) – Contains model inputs such that model(X\_sample) is a valid invocation of the model, used to valid model input type.

**Returns**

Nothing

**Return type**

None

**summary\_status()** → *DataFrame*

A summary table of the current status.

**Returns**

The summary stable of the current status.

**Return type***pd.DataFrame***update(\*\*kwargs)** → *GenericModel*

Updates model metadata in the Model Catalog. Updates only metadata information. The model artifacts are immutable and cannot be updated.

**Parameters****kwargs** –**display\_name:** (*str*, *optional*). **Defaults to None.**

The name of the model.

**description:** (*str*, *optional*). **Defaults to None.**

The description of the model.

**freeform\_tags**[*Dict(str, str)*, *Defaults to None.*] Freeform tags for the model.**defined\_tags**[(*Dict(str, dict(str, object))*), *optional*]. *Defaults to None.*] Defined tags for the model.

Additional kwargs arguments. Can be any attribute that *oci.data\_science.models.Model* accepts.

**Returns**An instance of *GenericModel* (self).**Return type***GenericModel***Raises****ValueError** – if model not saved to the Model Catalog.

**classmethod update\_deployment**(*model\_deployment\_id: Optional[str] = None*, *properties: Optional[Union[ModelDeploymentProperties, dict]] = None*, *wait\_for\_completion: bool = True*, *max\_wait\_time: int = 1200*, *poll\_interval: int = 30*, *\*\*kwargs*) → *ModelDeployment*

Updates a model deployment.

You can update *model\_deployment\_configuration\_details* and change *instance\_shape* and *model\_id* when the model deployment is in the ACTIVE lifecycle state. The *bandwidth\_mbps* or *instance\_count* can only

be updated while the model deployment is in the *INACTIVE* state. Changes to the *bandwidth\_mbps* or *instance\_count* will take effect the next time the *ActivateModelDeployment* action is invoked on the model deployment resource.

## Examples

```
>>> # Update access log id, freeform tags and description for the model_
    ↪ deployment
>>> model.update_deployment(
>>>     properties=ModelDeploymentProperties(
>>>         access_log_id=<log_ocid>,
>>>         description="Description for Custom Model",
>>>         freeform_tags={"key": "value"},
>>>     )
>>> )
```

## Parameters

- **model\_deployment\_id** (*str.*) – The model deployment OCID. Defaults to None. If the method called on instance level, then *self.model\_deployment.model\_deployment\_id* will be used.
- **properties** (*ModelDeploymentProperties* or *dict*) – The properties for updating the deployment.
- **wait\_for\_completion** (*bool*) – Flag set for whether to wait for deployment to complete before proceeding. Defaults to True.
- **max\_wait\_time** (*int*) – Maximum amount of time to wait in seconds (Defaults to 1200). Negative implies infinite wait time.
- **poll\_interval** (*int*) – Poll interval in seconds (Defaults to 60).
- **kwargs** –

**auth:** (*Dict*, optional). Defaults to *None*.

The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate *IdentityClient* object.

## Returns

An instance of *ModelDeployment* class.

## Return type

*ModelDeployment*

**upload\_artifact** (*uri: str, auth: Optional[Dict] = None, force\_overwrite: Optional[bool] = False*) → None

Uploads model artifacts to the provided *uri*. The artifacts will be zipped before uploading.

## Parameters

- **uri** (*str*) – The destination location for the model artifacts, which can be a local path or OCI object storage URI. Examples: `>>> upload_artifact(uri="/some/local/folder/") >>> upload_artifact(uri="oci://bucket@namespace/prefix/")`
- **auth** ((*Dict*, optional). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys*

or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate `IdentityClient` object.

- **force\_overwrite** (*bool*) – Overwrite `target_dir` if exists.

**verify**(*data: Optional[Any] = None, reload\_artifacts: bool = True, auto\_serialize\_data: bool = False, \*\*kwargs*) → Dict[str, Any]

Test if deployment works in local environment.

## Examples

```
>>> uri = "https://github.com/pytorch/hub/raw/master/images/dog.jpg"
>>> prediction = model.verify(image=uri)['prediction']
```

```
>>> # examples on storage options
>>> prediction = model.verify(
...     image="oci://<bucket>@<tenancy>/myimage.png",
...     storage_options=ads.auth.default_signer()
... )['prediction']
```

## Parameters

- **data** (*Any*) – Data used to test if deployment works in local environment.
- **reload\_artifacts** (*bool. Defaults to True.*) – Whether to reload artifacts or not.
- **auto\_serialize\_data** (*bool.*) – Whether to auto serialize input data. Defaults to *False* for `GenericModel`, and *True* for other frameworks. *data* required to be json serializable if *auto\_serialize\_data=False*. if *auto\_serialize\_data* set to *True*, data will be serialized before sending to model deployment endpoint.
- **kwargs** – `content_type`: str, used to indicate the media type of the resource. `image`: `PIL.Image` Object or uri for the image.

A valid string path for image file can be local path, http(s), oci, s3, gs.

### storage\_options: dict

Passed to `fsspec.open` for a particular storage connection. Please see `fsspec` (<https://filesystem-spec.readthedocs.io/en/latest/api.html#fsspec.open>) for more details.

## Returns

A dictionary which contains prediction results.

## Return type

Dict

```
class ads.model.generic_model.ModelState(value)
```

Bases: Enum

An enumeration.

```
AVAILABLE = 'Available'
```

```
DONE = 'Done'
```

```
NEEDSACTION = 'Needs Action'
```

**NOTAVAILABLE** = 'Not Available'

**exception** `ads.model.generic_model.NotActiveDeploymentError(state: str)`

Bases: `Exception`

**exception** `ads.model.generic_model.RuntimeInfoInconsistencyError`

Bases: `Exception`

**exception** `ads.model.generic_model.SerializeInputNotImplementedError`

Bases: `NotImplementedError`

**exception** `ads.model.generic_model.SerializeModelNotImplementedError`

Bases: `NotImplementedError`

**class** `ads.model.generic_model.SummaryStatus`

Bases: `object`

SummaryStatus class which track the status of the Model frameworks.

**update\_action**(*detail: str, action: str*) → `None`

Updates the action of the summary status table of the corresponding detail.

**Parameters**

- **detail** ((*str*)) – Value of the detail in the Details column. Used to locate which row to update.
- **status** ((*str*)) – New status to be updated for the row specified by detail.

**Returns**

Nothing.

**Return type**

`None`

**update\_status**(*detail: str, status: str*) → `None`

Updates the status of the summary status table of the corresponding detail.

**Parameters**

- **detail** ((*str*)) – value of the detail in the Details column. Used to locate which row to update.
- **status** ((*str*)) – new status to be updated for the row specified by detail.

**Returns**

Nothing.

**Return type**

`None`

#### 18.1.1.14.4 `ads.model.model_properties` module



```

class ads.model.model_properties.ModelProperties(inference_conda_env: Optional[str] = None,
                                                inference_python_version: Optional[str] = None,
                                                training_conda_env: Optional[str] = None,
                                                training_python_version: Optional[str] = None,
                                                training_resource_id: Optional[str] = None,
                                                training_script_path: Optional[str] = None,
                                                training_id: Optional[str] = None, compartment_id:
Optional[str] = None, project_id: Optional[str] =
None, bucket_uri: Optional[str] = None,
remove_existing_artifact: Optional[bool] = None,
overwrite_existing_artifact: Optional[bool] = None,
deployment_instance_shape: Optional[str] = None,
deployment_instance_count: Optional[int] = None,
deployment_bandwidth_mbps: Optional[int] =
None, deployment_log_group_id: Optional[str] =
None, deployment_access_log_id: Optional[str] =
None, deployment_predict_log_id: Optional[str] =
None, deployment_memory_in_gbs:
Optional[Union[float, int]] = None,
deployment_ocpus: Optional[Union[float, int]] =
None)

```

Bases: BaseProperties

Represents properties required to save and deploy model.

**bucket\_uri:** str = None

**compartment\_id:** str = None

**deployment\_access\_log\_id:** str = None

**deployment\_bandwidth\_mbps:** int = None

**deployment\_instance\_count:** int = None

**deployment\_instance\_shape:** str = None

**deployment\_log\_group\_id:** str = None

**deployment\_memory\_in\_gbs:** Union[float, int] = None

**deployment\_ocpus:** Union[float, int] = None

**deployment\_predict\_log\_id:** str = None

**inference\_conda\_env:** str = None

**inference\_python\_version:** str = None

**overwrite\_existing\_artifact:** bool = None

**project\_id:** str = None

**remove\_existing\_artifact:** bool = None

**training\_conda\_env:** str = None

**training\_id:** str = None

```
training_python_version: str = None

training_resource_id: str = None

training_script_path: str = None
```

#### 18.1.1.14.5 `ads.model.runtime.runtime_info` module

```
class ads.model.runtime.runtime_info.RuntimeInfo(model_artifact_version: str = "", model_deployment:
~ads.model.runtime.model_deployment_details.ModelDeploymentDetails, model_provenance:
= <factory>, model_provenance_details: ~ads.model.runtime.model_provenance_details.ModelProvenanceDetails,
= <factory>)
```

Bases: `DataClassSerializable`

`RuntimeInfo` class which is the data class representation of the runtime yaml file.

**classmethod** `from_env()` → *RuntimeInfo*

Populate the `RuntimeInfo` from environment variables.

**Returns**

A `RuntimeInfo` instance.

**Return type**

*RuntimeInfo*

```
model_artifact_version: str = ''
```

```
model_deployment: ModelDeploymentDetails
```

```
model_provenance: ModelProvenanceDetails
```

**save()**

Save the `RuntimeInfo` object into `runtime.yaml` file under the artifact directory.

**Returns**

Nothing.

**Return type**

None

#### 18.1.1.14.6 `ads.model.extractor.model_info_extractor_factory` module

```
class ads.model.extractor.model_info_extractor_factory.ModelInfoExtractorFactory
```

Bases: `object`

Class that extract Model Taxonomy Metadata for all supported frameworks.

**static** `extract_info(model)`

Extracts model taxonomy metadata.

**Parameters**

**model** (*[ADS model, sklearn, xgboost, lightgbm, keras, oracle\_automl]*) –  
The model object

**Returns**

A dictionary with keys of Framework, FrameworkVersion, Algorithm, Hyperparameters of the model

**Return type***ModelTaxonomyMetadata***Examples**

```
>>> from ads.common.model_info_extractor_factory import ModelInfoExtractorFactory
>>> metadata_taxonomy = ModelInfoExtractorFactory.extract_info(model)
```

**18.1.1.14.7 ads.model.extractor.model\_artifact module****18.1.1.14.8 ads.model.extractor.automl\_extractor module****class** ads.model.extractor.automl\_extractor.**AutoMLExtractor**(*model*)Bases: *ModelInfoExtractor*

Class that extract model metadata from automl models.

**model**

The model to extract metadata from.

**Type**

object

**estimator**

The estimator to extract metadata from.

**Type**

object

**property algorithm**

Extracts the algorithm of the model.

**Returns**

The algorithm of the model.

**Return type**

object

**property framework**

Extracts the framework of the model.

**Returns**

The framework of the model.

**Return type**

str

**property hyperparameter**

Extracts the hyperparameters of the model.

**Returns**

The hyperparameters of the model.

**Return type**

dict

**property version**

Extracts the framework version of the model.

**Returns**

The framework version of the model.

**Return type**

str

**18.1.1.14.9 ads.model.extractor.xgboost\_extractor module**

**class** ads.model.extractor.xgboost\_extractor.XgboostExtractor(*model*)

Bases: [\*ModelInfoExtractor\*](#)

Class that extract model metadata from xgboost models.

**model**

The model to extract metadata from.

**Type**

object

**estimator**

The estimator to extract metadata from.

**Type**

object

**framework**(*self*) → str

Returns the framework of the model.

**algorithm**(*self*) → object

Returns the algorithm of the model.

**version**(*self*) → str

Returns the version of framework of the model.

**hyperparameter**(*self*) → dict

Returns the hyperparameter of the model.

**property algorithm**

Extracts the algorithm of the model.

**Returns**

The algorithm of the model.

**Return type**

object

**property framework**

Extracts the framework of the model.

**Returns**

The framework of the model.

**Return type**

str

**property hyperparameter**

Extracts the hyperparameters of the model.

**Returns**

The hyperparameters of the model.

**Return type**

dict

**property version**

Extracts the framework version of the model.

**Returns**

The framework version of the model.

**Return type**

str

**18.1.1.14.10 ads.model.extractor.lightgbm\_extractor module**

**class** ads.model.extractor.lightgbm\_extractor.**LightgbmExtractor**(*model*)

Bases: [\*ModelInfoExtractor\*](#)

Class that extract model metadata from lightgbm models.

**model**

The model to extract metadata from.

**Type**

object

**estimator**

The estimator to extract metadata from.

**Type**

object

**framework**(*self*) → str

Returns the framework of the model.

**algorithm**(*self*) → object

Returns the algorithm of the model.

**version**(*self*) → str

Returns the version of framework of the model.

**hyperparameter**(*self*) → dict

Returns the hyperparameter of the model.

**property algorithm**

Extracts the algorithm of the model.

**Returns**

The algorithm of the model.

**Return type**

object

**property framework**

Extracts the framework of the model.

**Returns**

The framework of the model.

**Return type**

str

**property hyperparameter**

Extracts the hyperparameters of the model.

**Returns**

The hyperparameters of the model.

**Return type**

dict

**property version**

Extracts the framework version of the model.

**Returns**

The framework version of the model.

**Return type**

str

**18.1.1.14.11 ads.model.extractor.model\_info\_extractor module**

**class** ads.model.extractor.model\_info\_extractor.**ModelInfoExtractor**

Bases: ABC

The base abstract class to extract model metadata.

**framework**(*self*) → str

Returns the framework of the model.

**algorithm**(*self*) → object

Returns the algorithm of the model.

**version**(*self*) → str

Returns the version of framework of the model.

**hyperparameter**(*self*) → dict

Returns the hyperparameter of the model.

**info**(*self*) → dict

Returns the model taxonomy metadata information.

**abstract algorithm**()

The abstract method to extracts the algorithm of the model.

**Returns**

The algorithm of the model.

**Return type**

object

**abstract framework()**

The abstract method to extracts the framework of the model.

**Returns**

The framework of the model.

**Return type**

str

**abstract hyperparameter()**

The abstract method to extracts the hyperparameters of the model.

**Returns**

The hyperparameter of the model.

**Return type**

dict

**info()**

Extracts the taxonomy metadata of the model.

**Returns**

The taxonomy metadata of the model.

**Return type**

dict

**abstract version()**

The abstract method to extracts the framework version of the model.

**Returns**

The framework version of the model.

**Return type**

str

`ads.model.extractor.model_info_extractor.normalize_hyperparameter(data: Dict) → dict`

Converts all the fields to string to make sure it's json serializable.

**Parameters**

**data** (`(([Dict]))`) – The hyperparameter returned by the model.

**Returns**

Normalized (json serializable) dictionary.

**Return type**

Dict

**18.1.1.14.12 ads.model.extractor.sklearn\_extractor module**

**class** `ads.model.extractor.sklearn_extractor.SklearnExtractor(model)`

Bases: `ModelInfoExtractor`

Class that extract model metadata from sklearn models.

**model**

The model to extract metadata from.

**Type**

object

**estimator**

The estimator to extract metadata from.

**Type**

object

**framework**(*self*) → str

Returns the framework of the model.

**algorithm**(*self*) → object

Returns the algorithm of the model.

**version**(*self*) → str

Returns the version of framework of the model.

**hyperparameter**(*self*) → dict

Returns the hyperparameter of the model.

**property algorithm**

Extracts the algorithm of the model.

**Returns**

The algorithm of the model.

**Return type**

object

**property framework**

Extracts the framework of the model.

**Returns**

The framework of the model.

**Return type**

str

**property hyperparameter**

Extracts the hyperparameters of the model.

**Returns**

The hyperparameters of the model.

**Return type**

dict

**property version**

Extracts the framework version of the model.

**Returns**

The framework version of the model.

**Return type**

str



### 18.1.1.14.13 `ads.model.extractor.keras_extractor` module

**class** `ads.model.extractor.keras_extractor.KerasExtractor(model)`

Bases: [\*ModelInfoExtractor\*](#)

Class that extract model metadata from keras models.

**model**

The model to extract metadata from.

**Type**

object

**estimator**

The estimator to extract metadata from.

**Type**

object

**property algorithm**

Extracts the algorithm of the model.

**Returns**

The algorithm of the model.

**Return type**

object

**property framework**

Extracts the framework of the model.

**Returns**

The framework of the model.

**Return type**

str

**property hyperparameter**

Extracts the hyperparameters of the model.

**Returns**

The hyperparameters of the model.

**Return type**

dict

**property version**

Extracts the framework version of the model.

**Returns**

The framework version of the model.

**Return type**

str

#### 18.1.1.14.14 `ads.model.extractor.tensorflow_extractor` module

**class** `ads.model.extractor.tensorflow_extractor.TensorflowExtractor(model)`

Bases: [`ModelInfoExtractor`](#)

Class that extract model metadata from tensorflow models.

**model**

The model to extract metadata from.

**Type**

object

**estimator**

The estimator to extract metadata from.

**Type**

object

**framework**(*self*) → str

Returns the framework of the model.

**algorithm**(*self*) → object

Returns the algorithm of the model.

**version**(*self*) → str

Returns the version of framework of the model.

**hyperparameter**(*self*) → dict

Returns the hyperparameter of the model.

**property algorithm**

Extracts the algorithm of the model.

**Returns**

The algorithm of the model.

**Return type**

object

**property framework**

Extracts the framework of the model.

**Returns**

The framework of the model.

**Return type**

str

**property hyperparameter**

Extracts the hyperparameters of the model.

**Returns**

The hyperparameters of the model.

**Return type**

dict

**property version**

Extracts the framework version of the model.

**Returns**

The framework version of the model.

**Return type**

str

**18.1.1.14.15 ads.model.extractor.pytorch\_extractor module**

**class** ads.model.extractor.pytorch\_extractor.**PyTorchExtractor**(*model*)

Bases: [\*ModelInfoExtractor\*](#)

Class that extract model metadata from pytorch models.

**model**

The model to extract metadata from.

**Type**

object

**estimator**

The estimator to extract metadata from.

**Type**

object

**framework**(*self*) → str

Returns the framework of the model.

**algorithm**(*self*) → object

Returns the algorithm of the model.

**version**(*self*) → str

Returns the version of framework of the model.

**hyperparameter**(*self*) → dict

Returns the hyperparameter of the model.

**property algorithm**

Extracts the algorithm of the model.

**Returns**

The algorithm of the model.

**Return type**

object

**property framework**

Extracts the framework of the model.

**Returns**

The framework of the model.

**Return type**

str

**property hyperparameter**

Extracts the hyperparameters of the model.

**Returns**

The hyperparameters of the model.

**Return type**

dict

**property version**

Extracts the framework version of the model.

**Returns**

The framework version of the model.

**Return type**

str

**class** `ads.model.extractor.pytorch_extractor.PytorchExtractor(model)`

Bases: *PyTorchExtractor*

### 18.1.1.14.16 Module contents

### 18.1.1.15 `ads.model.deployment` package

#### 18.1.1.15.1 Submodules

#### 18.1.1.15.2 `ads.model.deployment.model_deployer` module

APIs to interact with Oracle's Model Deployment service.

There are three main classes: `ModelDeployment`, `ModelDeploymentDetails`, `ModelDeployer`.

One creates a `ModelDeployment` and deploys it under the umbrella of the `ModelDeployer` class. This way multiple `ModelDeployments` can be unified with one `ModelDeployer`. The `ModelDeployer` class also serves as the interface to all the deployments. `ModelDeploymentDetails` holds information about the particular details of a particular deployment, such as how many instances, etc. In this way multiple, independent `ModelDeployments` with the same details can be created using the `ModelDeployer` class.

### Examples

```
>>> from model_deploy.model_deployer import ModelDeployer, ModelDeploymentDetails
>>> deployer = ModelDeployer("model_dep_conf.yaml")
>>> deployment_properties = ModelDeploymentProperties(
...     'ocid1.datasciencemodel.ocn.reg.aaaaaaaaaaaaaaaaaaaaaaaa')
...     .with_prop('display_name', "My model display name")
...     .with_prop("project_id", project_id)
...     .with_prop("compartment_id", compartment_id)
...     .with_instance_configuration(
...         config={"INSTANCE_SHAPE": "VM.Standard.E3.Flex",
...                 "INSTANCE_COUNT": "1",
...                 "bandwidth_mbps": 10,
...                 "memory_in_gbs": 10,
...                 "ocpus": 2})
```

(continues on next page)

(continued from previous page)

```

...         ).build()
>>> deployment_info = deployer.deploy(deployment_properties,
...         max_wait_time=600, poll_interval=15)
>>> print(deployment_info.model_deployment_id)
>>> print(deployment_info.workflow_req_id)
>>> print(deployment_info.url)
>>> deployer.list_deployments() # Optionally pass in a status

```

**class** `ads.model.deployment.model_deployer.ModelDeployer` (*config*: *Optional[dict] = None*, *ds\_client*: *Optional[DataScienceClient] = None*)

Bases: `object`

ModelDeployer is the class responsible for deploying the ModelDeployment

**config**

ADS auth dictionary for OCI authentication.

**Type**

`dict`

**ds\_client**

data science client

**Type**

`DataScienceClient`

**ds\_composite\_client**

composite data science client

**Type**

`DataScienceCompositeClient`

**deploy**(*model\_deployment\_details*, *\\*\*kwargs*)

Deploy the model specified by *model\_deployment\_details*.

**get\_model\_deployment**(*model\_deployment\_id*: *str*)

Get the ModelDeployment specified by *model\_deployment\_id*.

**get\_model\_deployment\_state**(*model\_deployment\_id*)

Get the state of the current deployment specified by id.

**delete**(*model\_deployment\_id*, *\\*\*kwargs*)

Remove the model deployment specified by the id or Model Deployment Object

**list\_deployments**(*status*)

lists the model deployments associated with current compartment and data science client

**show\_deployments**(*status*)

shows the deployments filtered by *status* in a Dataframe

Initializes model deployer.

**Parameters**

**config** (*dict*, *optional*) –

ADS auth dictionary for OCI authentication.

This can be generated by calling `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()`. If this is `None`, `ads.common.default_signer(client_kwargs)` will be used.

**ds\_client:** `oci.data_science.data_science_client.DataScienceClient`

The Oracle DataScience client.

**delete**(*model\_deployment\_id*, *wait\_for\_completion*: *bool* = *True*, *max\_wait\_time*: *int* = *1200*, *poll\_interval*: *int* = *30*) → *ModelDeployment*

Deletes the model deployment specified by OCID.

#### Parameters

- **model\_deployment\_id** (*str*) – Model deployment OCID.
- **wait\_for\_completion** (*bool*) – Wait for deletion to complete. Defaults to *True*.
- **max\_wait\_time** (*int*) – Maximum amount of time to wait in seconds (Defaults to 600). Negative implies infinite wait time.
- **poll\_interval** (*int*) – Poll interval in seconds (Defaults to 60).

#### Return type

A *ModelDeployment* instance that was deleted

**deploy**(*properties*: *Optional*[*Union*[*ModelDeploymentProperties*, *Dict*]] = *None*, *wait\_for\_completion*: *bool* = *True*, *max\_wait\_time*: *int* = *1200*, *poll\_interval*: *int* = *30*, *\*\*kwargs*) → *ModelDeployment*

Deploys a model.

#### Parameters

- **properties** (*ModelDeploymentProperties* or *dict*) – Properties to deploy the model. Properties can be *None* when *kwargs* are used for specifying properties.
- **wait\_for\_completion** (*bool*) – Flag set for whether to wait for deployment to complete before proceeding. Optional, defaults to *True*.
- **max\_wait\_time** (*int*) – Maximum amount of time to wait in seconds. Optional, defaults to 1200. Negative value implies infinite wait time.
- **poll\_interval** (*int*) – Poll interval in seconds. Optional, defaults to 30.
- **kwargs** – Keyword arguments for initializing *ModelDeploymentProperties*. See *ModelDeploymentProperties()* for details.

#### Returns

A *ModelDeployment* instance.

#### Return type

*ModelDeployment*

**deploy\_from\_model\_uri**(*model\_uri*: *str*, *properties*: *Optional*[*Union*[*ModelDeploymentProperties*, *Dict*]] = *None*, *wait\_for\_completion*: *bool* = *True*, *max\_wait\_time*: *int* = *1200*, *poll\_interval*: *int* = *30*, *\*\*kwargs*) → *ModelDeployment*

Deploys a model.

#### Parameters

- **model\_uri** (*str*) – uri to model files, can be local or in cloud storage
- **properties** (*ModelDeploymentProperties* or *dict*) – Properties to deploy the model. Properties can be *None* when *kwargs* are used for specifying properties.
- **wait\_for\_completion** (*bool*) – Flag set for whether to wait for deployment to complete before proceeding. Defaults to *True*
- **max\_wait\_time** (*int*) – Maximum amount of time to wait in seconds (Defaults to 1200). Negative implies infinite wait time.

- **poll\_interval** (*int*) – Poll interval in seconds (Defaults to 30).
- **kwargs** – Keyword arguments for initializing ModelDeploymentProperties

**Returns**

A ModelDeployment instance

**Return type**

*ModelDeployment*

**get\_model\_deployment**(*model\_deployment\_id: str*) → *ModelDeployment*

Gets a ModelDeployment by OCID.

**Parameters**

**model\_deployment\_id** (*str*) – Model deployment OCID

**Returns**

A ModelDeployment instance

**Return type**

*ModelDeployment*

**get\_model\_deployment\_state**(*model\_deployment\_id: str*) → *State*

Gets the state of a deployment specified by OCID

**Parameters**

**model\_deployment\_id** (*str*) – Model deployment OCID

**Returns**

The state of the deployment

**Return type**

*str*

**list\_deployments**(*status=None, compartment\_id=None, \*\*kwargs*) → *list*

Lists the model deployments associated with current compartment and data science client

**Parameters**

- **status** (*str*) – Status of deployment. Defaults to None.
- **compartment\_id** (*str*) – Target compartment to list deployments from. Defaults to the compartment set in the environment variable “NB\_SESSION\_COMPARTMENT\_OCID”. If “NB\_SESSION\_COMPARTMENT\_OCID” is not set, the root compartment ID will be used. An ValueError will be raised if root compartment ID cannot be determined.
- **kwargs** – The values are passed to `oci.data_science.DataScienceClient.list_model_deployments`.

**Returns**

A list of ModelDeployment objects.

**Return type**

*list*

**Raises**

**ValueError** – If compartment\_id is not specified and cannot be determined from the environment.

**show\_deployments**(*status=None, compartment\_id=None*) → *DataFrame*

Returns the model deployments associated with current compartment and data science client as a Dataframe that can be easily visualized

**Parameters**

- **status** (*str*) – Status of deployment. Defaults to None.
- **compartment\_id** (*str*) – Target compartment to list deployments from. Defaults to the compartment set in the environment variable “NB\_SESSION\_COMPARTMENT\_OCID”. If “NB\_SESSION\_COMPARTMENT\_OCID” is not set, the root compartment ID will be used. An `ValueError` will be raised if root compartment ID cannot be determined.

**Returns**

pandas Dataframe containing information about the ModelDeployments

**Return type**

DataFrame

**Raises**

**ValueError** – If compartment\_id is not specified and cannot be determined from the environment.

**update**(*model\_deployment\_id: str, properties: Optional[ModelDeploymentProperties] = None, wait\_for\_completion: bool = True, max\_wait\_time: int = 1200, poll\_interval: int = 30, \*\*kwargs*) → *ModelDeployment*

Updates an existing model deployment.

**Parameters**

- **model\_deployment\_id** (*str*) – Model deployment OCID.
- **properties** (*ModelDeploymentProperties*) – An instance of *ModelDeploymentProperties* or dict to initialize the *ModelDeploymentProperties*. Defaults to None.
- **wait\_for\_completion** (*bool*) – Flag set for whether to wait for deployment to complete before proceeding. Defaults to True.
- **max\_wait\_time** (*int*) – Maximum amount of time to wait in seconds (Defaults to 1200).
- **poll\_interval** (*int*) – Poll interval in seconds (Defaults to 30).
- **kwargs** – Keyword arguments for initializing *ModelDeploymentProperties*.

**Returns**

A *ModelDeployment* instance

**Return type**

*ModelDeployment*

### 18.1.1.15.3 ads.model.deployment.model\_deployment module

**exception** `ads.model.deployment.model_deployment.LogNotConfiguredError`

Bases: `Exception`

**class** `ads.model.deployment.model_deployment.ModelDeployment`(*properties: Optional[Union[ModelDeploymentProperties, Dict]] = None, config: Optional[Dict] = None, workflow\_req\_id: Optional[str] = None, model\_deployment\_id: Optional[str] = None, model\_deployment\_url: str = "", \*\*kwargs*)

Bases: `object`

A class used to represent a Model Deployment.



**config**

Deployment configuration parameters

**Type**

(dict)

**properties**

ModelDeploymentProperties object

**Type**

(*ModelDeploymentProperties*)

**workflow\_state\_progress**

Workflow request id

**Type**

(str)

**workflow\_steps**

The number of steps in the workflow

**Type**

(int)

**url**

The model deployment url endpoint

**Type**

(str)

**ds\_client**

The data science client used by model deployment

**Type**

(DataScienceClient)

**ds\_composite\_client**

The composite data science client used by the model deployment

**Type**

(DataScienceCompositeClient)

**workflow\_req\_id**

Workflow request id

**Type**

(str)

**model\_deployment\_id**

model deployment id

**Type**

(str)

**state**

Returns the deployment state of the current Model Deployment object

**Type**

(*State*)

**deploy**(*wait\_for\_completion*, *\\*\*kwargs*)

Deploy the current Model Deployment object

**delete**(*wait\_for\_completion*, *\\*\*kwargs*)

Deletes the current Model Deployment object

**update**(*wait\_for\_completion*, *\\*\*kwargs*)

Updates a model deployment

**list\_workflow\_logs**()

Returns a list of the steps involved in deploying a model

Initializes a ModelDeployment object.

#### Parameters

- **properties** ((*Union[ModelDeploymentProperties, Dict]*, *optional*)). Defaults to *None*.) – Object containing deployment properties. The properties can be *None* when *kwargs* are used for specifying properties.
- **config** ((*Dict*, *optional*)). Defaults to *None*.) – ADS auth dictionary for OCI authentication. This can be generated by calling *ads.common.auth.api\_keys()* or *ads.common.auth.resource\_principal()*. If this is *None* then the *ads.common.default\_signer(client\_kwargs)* will be used.
- **workflow\_req\_id** ((*str*, *optional*)). Defaults to *None*.) – Workflow request id.
- **model\_deployment\_id** ((*str*, *optional*)). Defaults to *None*.) – Model deployment OCID.
- **model\_deployment\_url** ((*str*, *optional*)). Defaults to *empty string*.) – Model deployment url.
- **kwargs** – Keyword arguments for initializing *ModelDeploymentProperties*.

**property access\_log: OCILog**

Gets the model deployment access logs object.

#### Returns

The OCILog object containing the access logs.

#### Return type

OCILog

**delete**(*wait\_for\_completion: bool = True*, *max\_wait\_time: int = 1200*, *poll\_interval: int = 30*)

Deletes the ModelDeployment

#### Parameters

- **wait\_for\_completion** (*bool*) – Flag set for whether to wait for deployment to complete before proceeding. Defaults to *True*.
- **max\_wait\_time** (*int*) – Maximum amount of time to wait in seconds (Defaults to 600). Negative implies infinite wait time.
- **poll\_interval** (*int*) – Poll interval in seconds (Defaults to 60).

#### Returns

The instance of ModelDeployment.

#### Return type

*ModelDeployment*

**deploy**(*wait\_for\_completion: bool = True, max\_wait\_time: int = 1200, poll\_interval: int = 30*)

deploy deploys the current ModelDeployment object

#### Parameters

- **wait\_for\_completion** (*bool*) – Flag set for whether to wait for deployment to complete before proceeding. Defaults to True.
- **max\_wait\_time** (*int*) – Maximum amount of time to wait in seconds (Defaults to 600). Negative implies infinite wait time.
- **poll\_interval** (*int*) – Poll interval in seconds (Defaults to 60).

#### Returns

The instance of ModelDeployment.

#### Return type

*ModelDeployment*

**list\_workflow\_logs**() → list

Returns a list of the steps involved in deploying a model

#### Returns

List of dictionaries detailing the status of each step in the deployment process.

#### Return type

list

**logs**(*log\_type: Optional[str] = None*) → ConsolidatedLog

Gets the access or predict logs.

#### Parameters

**log\_type** ((*str, optional*)). Defaults to None.) – The log type. Can be “access”, “predict” or None.

#### Returns

The ConsolidatedLog object containing the logs.

#### Return type

ConsolidatedLog

**predict**(*json\_input=None, data: Optional[Any] = None, auto\_serialize\_data: bool = False, \*\*kwargs*) → dict

Returns prediction of input data run against the model deployment endpoint

#### Parameters

- **json\_input** (*Json serializable*) – Json payload for the prediction.
- **data** (*Any*) – Data for the prediction.
- **auto\_serialize\_data** (*bool*.) – Whether to auto serialize input data. Defaults to *False*. If *auto\_serialize\_data=False*, *data* required to be bytes or json serializable and *json\_input* required to be json serializable. If *auto\_serialize\_data* set to True, data will be serialized before sending to model deployment endpoint.
- **kwargs** –

#### **content\_type: str**

Used to indicate the media type of the resource. By default, it will be *application/octet-stream* for bytes input and *application/json* otherwise. The content-type header will be set to this value when calling the model deployment endpoint.

**Returns**

Prediction results.

**Return type**

dict

**property predict\_log: OCILog**

Gets the model deployment predict logs object.

**Returns**

The OCILog object containing the predict logs.

**Return type**

OCILog

**show\_logs**(*time\_start: Optional[datetime] = None, time\_end: Optional[datetime] = None, limit: int = 100, log\_type: Optional[str] = None*)

Shows deployment logs as a pandas dataframe.

**Parameters**

- **time\_start** ((*datetime.datetime, optional*). Defaults to *None*.) – Starting date and time in RFC3339 format for retrieving logs. Defaults to *None*. Logs will be retrieved 14 days from now.
- **time\_end** ((*datetime.datetime, optional*). Defaults to *None*.) – Ending date and time in RFC3339 format for retrieving logs. Defaults to *None*. Logs will be retrieved until now.
- **limit** ((*int, optional*). Defaults to *100*.) – The maximum number of items to return.
- **log\_type** ((*str, optional*). Defaults to *None*.) – The log type. Can be “access”, “predict” or *None*.

**Return type**

A pandas DataFrame containing logs.

**property state: State**

Returns the deployment state of the current Model Deployment object

**property status: State**

Returns the deployment state of the current Model Deployment object

**update**(*properties: Optional[Union[ModelDeploymentProperties, dict]] = None, wait\_for\_completion: bool = True, max\_wait\_time: int = 1200, poll\_interval: int = 30, \*\*kwargs*)

Updates a model deployment

You can update *model\_deployment\_configuration\_details* and change *instance\_shape* and *model\_id* when the model deployment is in the *ACTIVE* lifecycle state. The *bandwidth\_mbps* or *instance\_count* can only be updated while the model deployment is in the *INACTIVE* state. Changes to the *bandwidth\_mbps* or *instance\_count* will take effect the next time the *ActivateModelDeployment* action is invoked on the model deployment resource.

**Parameters**

- **properties** (*ModelDeploymentProperties* or *dict*) – The properties for updating the deployment.
- **wait\_for\_completion** (*bool*) – Flag set for whether to wait for deployment to complete before proceeding. Defaults to *True*.

- **max\_wait\_time** (*int*) – Maximum amount of time to wait in seconds (Defaults to 1200). Negative implies infinite wait time.
- **poll\_interval** (*int*) – Poll interval in seconds (Defaults to 60).
- **kwargs** – dict

**Returns**

The instance of ModelDeployment.

**Return type**

*ModelDeployment*

```
class ads.model.deployment.model_deployment.ModelDeploymentLogType
```

Bases: object

**ACCESS** = 'access'

**PREDICT** = 'predict'

#### 18.1.1.15.4 ads.model.deployment.model\_deployment\_properties module

```
class ads.model.deployment.model_deployment_properties.ModelDeploymentProperties(model_id:
                                                                                   Op-
                                                                                   tional[str]
                                                                                   = None,
                                                                                   model_uri:
                                                                                   Op-
                                                                                   tional[str]
                                                                                   = None,
                                                                                   oci_model_deployment:
                                                                                   Op-
                                                                                   tional[Union[ModelDeployment-
                                                                                   Create-
                                                                                   ModelDe-
                                                                                   ployment-
                                                                                   Details,
                                                                                   Update-
                                                                                   ModelDe-
                                                                                   ployment-
                                                                                   Details,
                                                                                   Dict]] =
                                                                                   None,
                                                                                   config:
                                                                                   Op-
                                                                                   tional[dict]
                                                                                   = None,
                                                                                   **kwargs)
```

Bases: OCIDataScienceMixin, ModelDeployment

Represents the details for a model deployment

**swagger\_types**

The property names and the corresponding types of OCI ModelDeployment model.

**Type**

dict

**model\_id**

The model artifact OCID in model catalog.

**Type**

str

**model\_uri**

uri to model files, can be local or in cloud storage.

**Type**

str

**with\_prop**(*property\_name*, *value*)

Set the model deployment details *property\_name* attribute to *value*

**with\_instance\_configuration**(*config*)

Set the configuration of VM instance.

**with\_access\_log**(*log\_group\_id*, *log\_id*)

Config the access log with OCI logging service

**with\_predict\_log**(*log\_group\_id*, *log\_id*)

Config the predict log with OCI logging service

**build()**

Return an instance of `CreateModelDeploymentDetails` for creating the deployment.

Initialize a `ModelDeploymentProperties` object by specifying one of the followings:

**Parameters**

- **model\_id** ((*str*, *optional*). Defaults to *None*.) – Model Artifact OCID. The `model_id` must be specified either explicitly or as an attribute of the OCI object.
- **model\_uri** ((*str*, *optional*). Defaults to *None*.) – Uri to model files, can be local or in cloud storage.
- **oci\_model\_deployment** ((*Union*[`ModelDeployment`, `CreateModelDeploymentDetails`, `UpdateModelDeploymentDetails`, `Dict`], *optional*). Defaults to *None*.) – An OCI model or Dict containing model deployment details. The OCI model can be an instance of either `ModelDeployment`, `CreateModelDeploymentDetails` or `UpdateModelConfigurationDetails`.
- **config** ((*Dict*, *optional*). Defaults to *None*.) – ADS auth dictionary for OCI authentication. This can be generated by calling `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()`. If this is *None*, `ads.common.default_signer(client_kwargs)` will be used.
- **kwargs** – Users can also initialize the object by using keyword arguments. The following keyword arguments are supported by `oci.data_science.models.data_science_models.ModelDeployment`:
  - *display\_name*,
  - *description*,
  - *project\_id*,
  - *compartment\_id*,
  - *model\_deployment\_configuration\_details*,
  - *category\_log\_details*,

- *freeform\_tags*,
- *defined\_tags*.

If *display\_name* is not specified, a randomly generated easy to remember name will be generated, like ‘strange-spider-2022-08-17-23:55.02’.

ModelDeploymentProperties also supports the following additional keyword arguments:

- *instance\_shape*,
- *instance\_count*,
- *bandwidth\_mbps*,
- *access\_log\_group\_id*,
- *access\_log\_id*,
- *predict\_log\_group\_id*,
- *predict\_log\_id*,
- *memory\_in\_gbs*,
- *ocpus*.

These additional arguments will be saved into appropriate properties in the OCI model.

#### Raises

**ValueError** – *model\_id* is None AND not specified in *oci\_model\_deployment.model\_deployment\_configuration\_details.model\_configuration\_details*.

**build()** → CreateModelDeploymentDetails

Converts the deployment properties to OCI CreateModelDeploymentDetails object. Converts a model URI into a model OCID if user passed in a URI.

#### Returns

A CreateModelDeploymentDetails instance ready for OCI API.

#### Return type

CreateModelDeploymentDetails

```
sub_properties = ['instance_shape', 'instance_count', 'bandwidth_mbps',
'access_log_group_id', 'access_log_id', 'predict_log_group_id', 'predict_log_id',
'memory_in_gbs', 'ocpus']
```

**to\_oci\_model(oci\_model)**

Convert properties into an OCI data model

#### Parameters

**oci\_model** (*class*) – The class of OCI data model, e.g., *oci.data\_science\_models.CreateModelDeploymentDetails*

**to\_update\_deployment()** → UpdateModelDeploymentDetails

Converts the deployment properties to OCI UpdateModelDeploymentDetails object.

#### Returns

An UpdateModelDeploymentDetails instance ready for OCI API.

#### Return type

CreateModelDeploymentDetails

**with\_access\_log**(*log\_group\_id: str, log\_id: str*)

Adds access log config

**Parameters**

- **group\_id** (*str*) – Log group ID of OCI logging service
- **log\_id** (*str*) – Log ID of OCI logging service

**Returns**

self

**Return type**

*ModelDeploymentProperties*

**with\_category\_log**(*log\_type: str, group\_id: str, log\_id: str*)

Adds category log configuration

**Parameters**

- **log\_type** (*str*) – The type of logging to be configured. Must be “access” or “predict”
- **group\_id** (*str*) – Log group ID of OCI logging service
- **log\_id** (*str*) – Log ID of OCI logging service

**Returns**

self

**Return type**

*ModelDeploymentProperties*

**Raises**

**ValueError** – When log\_type is invalid

**with\_instance\_configuration**(*config*)

with\_instance\_configuration creates a ModelDeploymentDetails object with a specific config

**Parameters**

**config** (*dict*) – dictionary containing instance configuration about the deployment. The following keys are supported:

- instance\_shape: str,
- instance\_count: int,
- bandwidth\_mbps: int,
- memory\_in\_gbs: float,
- ocpus: float

The instance\_shape and instance\_count are required when creating a new deployment. They are optional when updating an existing deployment.

**Returns**

self

**Return type**

*ModelDeploymentProperties*

**with\_logging\_configuration**(*access\_log\_group\_id: str, access\_log\_id: str, predict\_log\_group\_id: Optional[str] = None, predict\_log\_id: Optional[str] = None*)

Adds OCI logging configurations for OCI logging service



**Parameters**

- **access\_log\_group\_id** (*str*) – Log group ID of OCI logging service for access log
- **access\_log\_id** (*str*) – Log ID of OCI logging service for access log
- **predict\_log\_group\_id** (*str*) – Log group ID of OCI logging service for predict log
- **predict\_log\_id** (*str*) – Log ID of OCI logging service for predict log

**Returns**

self

**Return type***ModelDeploymentProperties***with\_predict\_log**(*log\_group\_id: str, log\_id: str*)

Adds predict log config

**Parameters**

- **group\_id** (*str*) – Log group ID of OCI logging service
- **log\_id** (*str*) – Log ID of OCI logging service

**Returns**

self

**Return type***ModelDeploymentProperties***with\_prop**(*property\_name: str, value: Any*)Sets model deployment's *property\_name* attribute to *value***Parameters**

- **property\_name** (*str*) – Name of a model deployment property.
- **value** – New value for property attribute.

**Returns**

self

**Return type***ModelDeploymentProperties***18.1.1.15.5 Module contents****18.1.1.16 ads.model.framework package****18.1.1.16.1 Submodules****18.1.1.16.2 ads.model.framework.automl\_model module**

**class** `ads.model.framework.automl_model.AutoMLModel`(*estimator: Callable, artifact\_dir: str, properties: Optional[ModelProperties] = None, auth: Optional[Dict] = None, \*\*kwargs*)

Bases: *FrameworkSpecificModel*

AutoMLModel class for estimators from AutoML framework.

**algorithm**

“ensemble”, the algorithm name of the model.

**Type**

str

**artifact\_dir**

Artifact directory to store the files needed for deployment.

**Type**

str

**auth**

Default authentication is set using the *ads.set\_auth* API. To override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create an authentication signer to instantiate an IdentityClient object.

**Type**

Dict

**ds\_client**

The data science client used by model deployment.

**Type**

DataScienceClient

**estimator**

A trained automl estimator/model using oracle automl.

**Type**

Callable

**framework**

“oracle\_automl”, the framework name of the estimator.

**Type**

str

**hyperparameter**

The hyperparameters of the estimator.

**Type**

dict

**metadata\_custom**

The model custom metadata.

**Type**

ModelCustomMetadata

**metadata\_provenance**

The model provenance metadata.

**Type**

ModelProvenanceMetadata

**metadata\_taxonomy**

The model taxonomy metadata.

**Type**

ModelTaxonomyMetadata

**model\_artifact**

This is built by calling prepare.

**Type**

*ModelArtifact*

**model\_deployment**

A ModelDeployment instance.

**Type**

*ModelDeployment*

**model\_file\_name**

Name of the serialized model. Default to “model.pkl”.

**Type**

str

**model\_id**

The model ID.

**Type**

str

**properties**

ModelProperties object required to save and deploy model.

**Type**

*ModelProperties*

**runtime\_info**

A RuntimeInfo instance.

**Type**

*RuntimeInfo*

**schema\_input**

Schema describes the structure of the input data.

**Type**

Schema

**schema\_output**

Schema describes the structure of the output data.

**Type**

Schema

**serialize**

Whether to serialize the model to pkl file by default. If False, you need to serialize the model manually, save it under artifact\_dir and update the score.py manually.

**Type**

bool

**version**

The framework version of the model.

**Type**

str

**delete\_deployment(...)**

Deletes the current model deployment.

**deploy(..., \\*\*kwargs)**

Deploys a model.

**from\_model\_artifact(uri, model\_file\_name, artifact\_dir, ..., \\*\*kwargs)**

Loads model from the specified folder, or zip/tar archive.

**from\_model\_catalog(model\_id, model\_file\_name, artifact\_dir, ..., \\*\*kwargs)**

Loads model from model catalog.

**introspect(...)**

Runs model introspection.

**predict(data, ...)**

Returns prediction of input data run against the model deployment endpoint.

**prepare(..., \\*\*kwargs)**

Prepare and save the score.py, serialized model and runtime.yaml file.

**reload(...)**

Reloads the model artifact files: *score.py* and the *runtime.yaml*.

**save(..., \\*\*kwargs)**

Saves model artifacts to the model catalog.

**summary\_status(...)**

Gets a summary table of the current status.

**verify(data, ...)**

Tests if deployment works in local environment.

## Examples

```
>>> import tempfile
>>> import logging
>>> import warnings
>>> from ads.automl.driver import AutoML
>>> from ads.automl.provider import OracleAutoMLProvider
>>> from ads.dataset.dataset_browser import DatasetBrowser
>>> from ads.model.framework.automl_model import AutoMLModel
>>> from ads.common.model_metadata import UseCaseType
>>> ds = DatasetBrowser.sklearn().open("wine").set_target("target")
>>> train, test = ds.train_test_split(test_size=0.1, random_state = 42)
```

```
>>> ml_engine = OracleAutoMLProvider(n_jobs=-1, loglevel=logging.ERROR)
>>> oracle_automl = AutoML(train, provider=ml_engine)
>>> model, baseline = oracle_automl.train(
...     model_list=['LogisticRegression', 'DecisionTreeClassifier'],
...     random_state = 42,
...     time_budget = 500
... )
```

```
>>> automl_model.prepare(inference_conda_env=inference_conda_env, force_
↳ overwrite=True)
>>> automl_model.verify(...)
>>> automl_model.save()
>>> model_deployment = automl_model.deploy(wait_for_completion=False)
```

Initiates a AutoMLModel instance.

#### Parameters

- **estimator** (*Callable*) – Any model object generated by automl framework.
- **artifact\_dir** (*str*) – Directory for generate artifact.
- **properties** (*(ModelProperties, optional). Defaults to None.*) – ModelProperties object required to save and deploy model.
- **auth** (*(Dict, optional). Defaults to None.*) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

#### Returns

AutoMLModel instance.

#### Return type

*AutoMLModel*

#### Raises

**TypeError** – If the input model is not an AutoML model.

**serialize\_model** (*force\_overwrite: Optional[bool] = False, X\_sample: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, \*\*kwargs: Dict*)

Serialize and save AutoML model using pickle.

#### Parameters

- **force\_overwrite** (*(bool, optional). Defaults to False.*) – If set as True, overwrite serialized model if exists.
- **X\_sample** (*Union[Dict, str, List, np.ndarray, pd.core.series.Series, pd.core.frame.DataFrame,]. Defaults to None.*) – Contains model inputs such that *model(X\_sample)* is a valid invocation of the model. Used to generate input schema.

#### Returns

Nothing.

#### Return type

None

### 18.1.1.16.3 ads.model.framework.lightgbm\_model module

```
class ads.model.framework.lightgbm_model.LightGBMModel(estimator: Callable, artifact_dir: str,  
                                                       properties: Optional[ModelProperties] =  
                                                       None, auth: Optional[Dict] = None,  
                                                       **kwargs)
```

Bases: [FrameworkSpecificModel](#)

LightGBMModel class for estimators from Lightgbm framework.

**algorithm**

The algorithm of the model.

**Type**

str

**artifact\_dir**

Artifact directory to store the files needed for deployment.

**Type**

str

**auth**

Default authentication is set using the `ads.set_auth` API. To override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create an authentication signer to instantiate an IdentityClient object.

**Type**

Dict

**ds\_client**

The data science client used by model deployment.

**Type**

DataScienceClient

**estimator**

A trained lightgbm estimator/model using Lightgbm.

**Type**

Callable

**framework**

“lightgbm”, the framework name of the model.

**Type**

str

**hyperparameter**

The hyperparameters of the estimator.

**Type**

dict

**metadata\_custom**

The model custom metadata.

**Type**

ModelCustomMetadata

**metadata\_provenance**

The model provenance metadata.

**Type**

ModelProvenanceMetadata

**metadata\_taxonomy**

The model taxonomy metadata.

**Type**

ModelTaxonomyMetadata

**model\_artifact**

This is built by calling prepare.

**Type**

*ModelArtifact*

**model\_deployment**

A ModelDeployment instance.

**Type**

*ModelDeployment*

**model\_file\_name**

Name of the serialized model.

**Type**

str

**model\_id**

The model ID.

**Type**

str

**properties**

ModelProperties object required to save and deploy model.

**Type**

*ModelProperties*

**runtime\_info**

A RuntimeInfo instance.

**Type**

*RuntimeInfo*

**schema\_input**

Schema describes the structure of the input data.

**Type**

Schema

**schema\_output**

Schema describes the structure of the output data.

**Type**

Schema

**serialize**

Whether to serialize the model to pkl file by default. If False, you need to serialize the model manually, save it under `artifact_dir` and update the `score.py` manually.

**Type**

bool

**version**

The framework version of the model.

**Type**

str

**delete\_deployment(...)**

Deletes the current model deployment.

**deploy(..., \*\*kwargs)**

Deploys a model.

**from\_model\_artifact(uri, model\_file\_name, artifact\_dir, ..., \*\*kwargs)**

Loads model from the specified folder, or zip/tar archive.

**from\_model\_catalog(model\_id, model\_file\_name, artifact\_dir, ..., \*\*kwargs)**

Loads model from model catalog.

**introspect(...)**

Runs model introspection.

**predict(data, ...)**

Returns prediction of input data run against the model deployment endpoint.

**prepare(..., \*\*kwargs)**

Prepare and save the `score.py`, serialized model and `runtime.yaml` file.

**reload(...)**

Reloads the model artifact files: `score.py` and the `runtime.yaml`.

**save(..., \*\*kwargs)**

Saves model artifacts to the model catalog.

**summary\_status(...)**

Gets a summary table of the current status.

**verify(data, ...)**

Tests if deployment works in local environment.

**Examples**

```
>>> import lightgbm as lgb
>>> import tempfile
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.datasets import load_iris
>>> from ads.model.framework.lightgbm_model import LightGBMModel
```

```
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
```



```
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
>>> train = lgb.Dataset(X_train, label=y_train)
>>> param = {
...     'objective': 'multiclass', 'num_class': 3,
... }
>>> lightgbm_estimator = lgb.train(param, train)
```

```
>>> lightgbm_model = LightGBMModel(estimator=lightgbm_estimator,
... artifact_dir=tempfile.mkdtemp())
```

```
>>> lightgbm_model.prepare(inference_conda_env="generalml_p37_cpu_v1", force_
↳ overwrite=True)
>>> lightgbm_model.reload()
>>> lightgbm_model.verify(X_test)
>>> lightgbm_model.save()
>>> model_deployment = lightgbm_model.deploy(wait_for_completion=False)
>>> lightgbm_model.predict(X_test)
```

Initiates a LightGBMModel instance. This class wraps the Lightgbm model as estimator. It's primary purpose is to hold the trained model and do serialization.

#### Parameters

- **estimator** – any model object generated by Lightgbm framework
- **artifact\_dir** (*str*) – Directory for generate artifact.
- **properties** (*(ModelProperties, optional). Defaults to None.*) – ModelProperties object required to save and deploy model.
- **auth** (*(Dict, optional). Defaults to None.*) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

#### Returns

LightGBMModel instance.

#### Return type

*LightGBMModel*

#### Raises

**TypeError** – If the input model is not a Lightgbm model or not supported for serialization.:

#### Examples

```
>>> import lightgbm as lgb
>>> import tempfile
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.datasets import load_iris
>>> from ads.model.framework.lightgbm_model import LightGBMModel
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
>>> train = lgb.Dataset(X_train, label=y_train)
```

(continues on next page)

(continued from previous page)

```

>>> param = {
...     'objective': 'multiclass', 'num_class': 3,
... }
>>> lightgbm_estimator = lgb.train(param, train)
>>> lightgbm_model = LightGBMModel(estimator=lightgbm_estimator, artifact_
    ↪dir=tempfile.mkdtemp())
>>> lightgbm_model.prepare(inference_conda_env="generalml_p37_cpu_v1")
>>> lightgbm_model.verify(X_test)
>>> lightgbm_model.save()
>>> model_deployment = lightgbm_model.deploy()
>>> lightgbm_model.predict(X_test)
>>> lightgbm_model.delete_deployment()

```

**generate\_initial\_types**(*X\_sample*: Any) → List

Auto generate initial types.

**Parameters**

**X\_sample** ((Any)) – Train data.

**Returns**

Initial types.

**Return type**

List

**serialize\_model**(*as\_onnx*: bool = False, *initial\_types*: Optional[List[Tuple]] = None, *force\_overwrite*: bool = False, *X\_sample*: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, \*\*kwargs: Dict)

Serialize and save Lightgbm model using ONNX or model specific method.

**Parameters**

- **artifact\_dir** (str) – Directory for generate artifact.
- **as\_onnx** ((boolean, optional). Defaults to False.) – If set as True, provide initial\_types or X\_sample to convert into ONNX.
- **initial\_types** ((List[Tuple], optional). Defaults to None.) – Each element is a tuple of a variable name and a type.
- **force\_overwrite** ((boolean, optional). Defaults to False.) – If set as True, overwrite serialized model if exists.
- **X\_sample** (Union[Dict, str, List, np.ndarray, pd.core.series.Series, pd.core.frame.DataFrame,]). Defaults to None.) – Contains model inputs such that model(X\_sample) is a valid invocation of the model. Used to generate initial\_types.

**Returns**

Nothing.

**Return type**

None

**to\_onnx**(*initial\_types*: List[Tuple] = None, *X\_sample*: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, \*\*kwargs)

Produces an equivalent ONNX model of the given Lightgbm model.

**Parameters**

- **initial\_types** (*List[Tuple], optional*). Defaults to *None*.) – Each element is a tuple of a variable name and a type.
- **X\_sample** (*Union[Dict, str, List, np.ndarray, pd.core.series.Series, pd.core.frame.DataFrame,]*. Defaults to *None*.) – Contains model inputs such that `model(X_sample)` is a valid invocation of the model. Used to generate `initial_types`.

**Returns**

An ONNX model (type

**Return type**

`ModelProto`) which is equivalent to the input Lightgbm model.

#### 18.1.1.16.4 `ads.model.framework.pytorch_model` module

```
class ads.model.framework.pytorch_model.PyTorchModel(estimator: callable, artifact_dir: str,  
                                                    properties: Optional[ModelProperties] =  
                                                    None, auth: Dict = None, **kwargs)
```

Bases: `FrameworkSpecificModel`

PyTorchModel class for estimators from Pytorch framework.

**algorithm**

The algorithm of the model.

**Type**

str

**artifact\_dir**

Artifact directory to store the files needed for deployment.

**Type**

str

**auth**

Default authentication is set using the `ads.set_auth` API. To override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create an authentication signer to instantiate an `IdentityClient` object.

**Type**

Dict

**ds\_client**

The data science client used by model deployment.

**Type**

`DataScienceClient`

**estimator**

A trained pytorch estimator/model using Pytorch.

**Type**

Callable

**framework**

“pytorch”, the framework name of the model.

**Type**

str

**hyperparameter**

The hyperparameters of the estimator.

**Type**

dict

**metadata\_custom**

The model custom metadata.

**Type**

ModelCustomMetadata

**metadata\_provenance**

The model provenance metadata.

**Type**

ModelProvenanceMetadata

**metadata\_taxonomy**

The model taxonomy metadata.

**Type**

ModelTaxonomyMetadata

**model\_artifact**

This is built by calling prepare.

**Type**

*ModelArtifact*

**model\_deployment**

A ModelDeployment instance.

**Type**

*ModelDeployment*

**model\_file\_name**

Name of the serialized model.

**Type**

str

**model\_id**

The model ID.

**Type**

str

**properties**

ModelProperties object required to save and deploy model.

**Type**

*ModelProperties*

**runtime\_info**

A RuntimeInfo instance.

**Type**

*RuntimeInfo*

**schema\_input**

Schema describes the structure of the input data.

**Type**

Schema

**schema\_output**

Schema describes the structure of the output data.

**Type**

Schema

**serialize**

Whether to serialize the model to pkl file by default. If False, you need to serialize the model manually, save it under `artifact_dir` and update the `score.py` manually.

**Type**

bool

**version**

The framework version of the model.

**Type**

str

**delete\_deployment(...)**

Deletes the current model deployment.

**deploy(..., *\*\*kwargs*)**

Deploys a model.

**from\_model\_artifact(uri, model\_file\_name, artifact\_dir, ..., *\*\*kwargs*)**

Loads model from the specified folder, or zip/tar archive.

**from\_model\_catalog(model\_id, model\_file\_name, artifact\_dir, ..., *\*\*kwargs*)**

Loads model from model catalog.

**introspect(...)**

Runs model introspection.

**predict(data, ...)**

Returns prediction of input data run against the model deployment endpoint.

**prepare(..., *\*\*kwargs*)**

Prepare and save the `score.py`, serialized model and `runtime.yaml` file.

**reload(...)**

Reloads the model artifact files: `score.py` and the `runtime.yaml`.

**save(..., *\*\*kwargs*)**

Saves model artifacts to the model catalog.

**summary\_status(...)**

Gets a summary table of the current status.

**verify(data, ...)**

Tests if deployment works in local environment.

## Examples

```
>>> torch_model = PyTorchModel(estimator=torch_estimator,
... artifact_dir=tmp_model_dir)
>>> inference_conda_env = "generalml_p37_cpu_v1"
```

```
>>> torch_model.prepare(inference_conda_env=inference_conda_env, force_
↳ overwrite=True)
>>> torch_model.reload()
>>> torch_model.verify(...)
>>> torch_model.save()
>>> model_deployment = torch_model.deploy(wait_for_completion=False)
>>> torch_model.predict(...)
```

Initiates a PyTorchModel instance.

### Parameters

- **estimator** (*callable*) – Any model object generated by pytorch framework
- **artifact\_dir** (*str*) – artifact directory to store the files needed for deployment.
- **properties** (*(ModelProperties, optional). Defaults to None.*) – ModelProperties object required to save and deploy model.
- **auth** (*(Dict, optional). Defaults to None.*) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

### Returns

PyTorchModel instance.

### Return type

*PyTorchModel*

**get\_data\_serializer**(*data: Union[Dict, str, List, ndarray, Series, DataFrame, torch.Tensor], data\_type: str = None*)

Returns serializable input data.

### Parameters

- **data** (*Union[Dict, str, list, numpy.ndarray, pd.core.series.Series,)* –
- **pd.core.frame.DataFrame** – Data expected by the model deployment predict API.
- **torch.Tensor]** – Data expected by the model deployment predict API.
- **data\_type** (*str*) – Type of the data.

### Returns

A class containing serialized input data and original data type information.

### Return type

InputDataSerializer

### Raises

**TypeError** – if provided data type is not supported.

**serialize\_model**(*as\_onnx: bool = False, force\_overwrite: bool = False, X\_sample: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, use\_torch\_script: bool = None, \*\*kwargs*) → None

Serialize and save Pytorch model using ONNX or model specific method.

#### Parameters

- **as\_onnx** ((*bool*, *optional*). Defaults to *False*.) – If set as *True*, convert into ONNX model.
- **force\_overwrite** ((*bool*, *optional*). Defaults to *False*.) – If set as *True*, overwrite serialized model if exists.
- **X\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]*. Defaults to *None*.) – A sample of input data that will be used to generate input schema and detect onnx\_args.
- **use\_torch\_script** ((*bool*, *optional*). Defaults to *None* (If the default value has not been changed, it will be set as *False*.) – If set as *True*, the model will be serialized as a TorchScript program. Check [https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html#export-load-model-in-torchscript-format](https://pytorch.org/tutorials/beginner/saving_loading_models.html#export-load-model-in-torchscript-format) for more details. If set as *False*, it will only save the trained model's learned parameters, and the score.py need to be modified to construct the model class instance first. Check [https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html#save-load-state-dict-recommended](https://pytorch.org/tutorials/beginner/saving_loading_models.html#save-load-state-dict-recommended) for more details.
- **\*\*kwargs** (*optional params used to serialize pytorch model to onnx*), –
- **following** (*including the*) – onnx\_args: (tuple or torch.Tensor), default to *None* Contains model inputs such that model(onnx\_args) is a valid invocation of the model. Can be structured either as: 1) ONLY A TUPLE OF ARGUMENTS; 2) A TENSOR; 3) A TUPLE OF ARGUMENTS ENDING WITH A DICTIONARY OF NAMED ARGUMENTS  
input\_names: (*List[str]*, *optional*). Names to assign to the input nodes of the graph, in order.  
output\_names: (*List[str]*, *optional*). Names to assign to the output nodes of the graph, in order.  
dynamic\_axes: (*dict*, *optional*), default to *None*. Specify axes of tensors as dynamic (i.e. known only at run-time).

#### Returns

Nothing.

#### Return type

*None*

**to\_onnx**(*path: str = None, onnx\_args=None, X\_sample: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, input\_names: List[str] = ['input'], output\_names: List[str] = ['output'], dynamic\_axes=None*)

Exports the given Pytorch model into ONNX format.

#### Parameters

- **path** (*str*, default to *None*) – Path to save the serialized model.
- **onnx\_args** ((*tuple or torch.Tensor*), default to *None*) – Contains model inputs such that model(onnx\_args) is a valid invocation of the model. Can be structured either as: 1) ONLY A TUPLE OF ARGUMENTS; 2) A TENSOR; 3) A TUPLE OF ARGUMENTS ENDING WITH A DICTIONARY OF NAMED ARGUMENTS
- **X\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]*. Defaults to *None*.) – A sample of input data that will be used to generate input schema and detect onnx\_args.
- **input\_names** ((*List[str]*, *optional*). Defaults to *["input"]*.) – Names to assign to the input nodes of the graph, in order.

- **output\_names** ((*List[str]*, *optional*). Defaults to ["output"].) – Names to assign to the output nodes of the graph, in order.
- **dynamic\_axes** ((*dict*, *optional*). Defaults to *None*.) – Specify axes of tensors as dynamic (i.e. known only at run-time).

**Returns**

Nothing

**Return type**

None

**Raises**

- **AssertionError** – if onnx module is not support by the current version of torch
- **ValueError** – if X\_sample is not provided if path is not provided

#### 18.1.1.16.5 ads.model.framework.sklearn\_model module

```
class ads.model.framework.sklearn_model.SklearnModel(estimator: Callable, artifact_dir: str,  
                                                    properties: Optional[ModelProperties] =  
                                                    None, auth: Optional[Dict] = None, **kwargs)
```

Bases: *FrameworkSpecificModel*

SklearnModel class for estimators from sklearn framework.

**algorithm**

The algorithm of the model.

**Type**

str

**artifact\_dir**

Artifact directory to store the files needed for deployment.

**Type**

str

**auth**

Default authentication is set using the *ads.set\_auth* API. To override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create an authentication signer to instantiate an IdentityClient object.

**Type**

Dict

**ds\_client**

The data science client used by model deployment.

**Type**

DataScienceClient

**estimator**

A trained sklearn estimator/model using scikit-learn.

**Type**

Callable



**framework**

“scikit-learn”, the framework name of the model.

**Type**

str

**hyperparameter**

The hyperparameters of the estimator.

**Type**

dict

**metadata\_custom**

The model custom metadata.

**Type**

ModelCustomMetadata

**metadata\_provenance**

The model provenance metadata.

**Type**

ModelProvenanceMetadata

**metadata\_taxonomy**

The model taxonomy metadata.

**Type**

ModelTaxonomyMetadata

**model\_artifact**

This is built by calling prepare.

**Type**

*ModelArtifact*

**model\_deployment**

A ModelDeployment instance.

**Type**

*ModelDeployment*

**model\_file\_name**

Name of the serialized model.

**Type**

str

**model\_id**

The model ID.

**Type**

str

**properties**

ModelProperties object required to save and deploy model.

**Type**

*ModelProperties*

**runtime\_info**

A RuntimeInfo instance.

**Type**

*RuntimeInfo*

**schema\_input**

Schema describes the structure of the input data.

**Type**

Schema

**schema\_output**

Schema describes the structure of the output data.

**Type**

Schema

**serialize**

Whether to serialize the model to pkl file by default. If False, you need to serialize the model manually, save it under artifact\_dir and update the score.py manually.

**Type**

bool

**version**

The framework version of the model.

**Type**

str

**delete\_deployment(...)**

Deletes the current model deployment.

**deploy(..., \\*\*kwargs)**

Deploys a model.

**from\_model\_artifact(uri, model\_file\_name, artifact\_dir, ..., \\*\*kwargs)**

Loads model from the specified folder, or zip/tar archive.

**from\_model\_catalog(model\_id, model\_file\_name, artifact\_dir, ..., \\*\*kwargs)**

Loads model from model catalog.

**introspect(...)**

Runs model introspection.

**predict(data, ...)**

Returns prediction of input data run against the model deployment endpoint.

**prepare(..., \\*\*kwargs)**

Prepare and save the score.py, serialized model and runtime.yaml file.

**reload(...)**

Reloads the model artifact files: *score.py* and the *runtime.yaml*.

**save(..., \\*\*kwargs)**

Saves model artifacts to the model catalog.

**summary\_status(...)**

Gets a summary table of the current status.

**verify**(*data*, ...)

Tests if deployment works in local environment.

## Examples

```
>>> import tempfile
>>> from sklearn.model_selection import train_test_split
>>> from ads.model.framework.sklearn_model import SklearnModel
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import load_iris
```

```
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
>>> sklearn_estimator = LogisticRegression()
>>> sklearn_estimator.fit(X_train, y_train)
```

```
>>> sklearn_model = SklearnModel(estimator=sklearn_estimator,
... artifact_dir=tmp_model_dir)
```

```
>>> sklearn_model.prepare(inference_conda_env="generalml_p37_cpu_v1", force_
↳ overwrite=True)
>>> sklearn_model.reload()
>>> sklearn_model.verify(X_test)
>>> sklearn_model.save()
>>> model_deployment = sklearn_model.deploy(wait_for_completion=False)
>>> sklearn_model.predict(X_test)
```

Initiates a SklearnModel instance.

### Parameters

- **estimator** (*Callable*) – Sklearn Model
- **artifact\_dir** (*str*) – Directory for generate artifact.
- **properties** (*(ModelProperties, optional)*). Defaults to *None*.) – ModelProperties object required to save and deploy model.
- **auth** (*(Dict, optional)*). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

### Returns

SklearnModel instance.

### Return type

*SklearnModel*

## Examples

```
>>> import tempfile
>>> from sklearn.model_selection import train_test_split
>>> from ads.framework.sklearn_model import SklearnModel
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import load_iris
```

```
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
>>> sklearn_estimator = LogisticRegression()
>>> sklearn_estimator.fit(X_train, y_train)
```

```
>>> sklearn_model = SklearnModel(estimator=sklearn_estimator, artifact_dir=tempfile.
↳ makedirs())
>>> sklearn_model.prepare(inference_conda_env="dataexpl_p37_cpu_v3")
>>> sklearn_model.verify(X_test)
>>> sklearn_model.save()
>>> model_deployment = sklearn_model.deploy()
>>> sklearn_model.predict(X_test)
>>> sklearn_model.delete_deployment()
```

**generate\_initial\_types**(*X\_sample: Any*) → List

Auto generate initial types.

### Parameters

**X\_sample** ((*Any*)) – Train data.

### Returns

Initial types.

### Return type

List

**static is\_either\_numerical\_or\_string\_dataframe**(*data: DataFrame*) → bool

Check whether all the columns are either numerical or string for dataframe.

**serialize\_model**(*as\_onnx: Optional[bool] = False, initial\_types: Optional[List[Tuple]] = None, force\_overwrite: Optional[bool] = False, X\_sample: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, \*\*kwargs: Dict*)

Serialize and save scikit-learn model using ONNX or model specific method.

### Parameters

- **as\_onnx** ((*bool, optional*)). Defaults to *False*.) – If set as *True*, provide initial\_types or *X\_sample* to convert into ONNX.
- **initial\_types** ((*List[Tuple]*, *optional*)). Defaults to *None*.) – Each element is a tuple of a variable name and a type.
- **force\_overwrite** ((*bool, optional*)). Defaults to *False*.) – If set as *True*, overwrite serialized model if exists.
- **X\_sample** (*Union[Dict, str, List, np.ndarray, pd.core.series.Series, pd.core.frame.DataFrame,]*). Defaults to *None*.) – Contains model inputs such that *model(X\_sample)* is a valid invocation of the model. Used to generate initial\_types.

**Returns**

Nothing.

**Return type**

None

**to\_onnx**(*initial\_types*: *List[Tuple] = None*, *X\_sample*: *Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None*, *\*\*kwargs*)

Produces an equivalent ONNX model of the given scikit-learn model.

**Parameters**

- **initial\_types** (*List[Tuple]*, *optional*). *Defaults to None.*) – Each element is a tuple of a variable name and a type.
- **X\_sample** (*Union[Dict, str, List, np.ndarray, pd.core.series.Series, pd.core.frame.DataFrame,]*. *Defaults to None.*) – Contains model inputs such that `model(X_sample)` is a valid invocation of the model. Used to generate *initial\_types*.

**Returns**

An ONNX model (type: `ModelProto`) which is equivalent to the input scikit-learn model.

**Return type**

`onnx.onnx_ml_pb2.ModelProto`

#### 18.1.1.16.6 `ads.model.framework.tensorflow_model` module

**class** `ads.model.framework.tensorflow_model.TensorFlowModel`(*estimator*: *callable*, *artifact\_dir*: *str*, *properties*: *Optional[ModelProperties]* = *None*, *auth*: *Dict = None*, *\*\*kwargs*)

Bases: `FrameworkSpecificModel`

TensorFlowModel class for estimators from Tensorflow framework.

**algorithm**

The algorithm of the model.

**Type**

`str`

**artifact\_dir**

Directory for generate artifact.

**Type**

`str`

**auth**

Default authentication is set using the `ads.set_auth` API. To override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create an authentication signer to instantiate an `IdentityClient` object.

**Type**

`Dict`

**ds\_client**

The data science client used by model deployment.

**Type**

`DataScienceClient`

**estimator**

A trained tensorflow estimator/model using Tensorflow.

**Type**

Callable

**framework**

“tensorflow”, the framework name of the model.

**Type**

str

**hyperparameter**

The hyperparameters of the estimator.

**Type**

dict

**metadata\_custom**

The model custom metadata.

**Type**

ModelCustomMetadata

**metadata\_provenance**

The model provenance metadata.

**Type**

ModelProvenanceMetadata

**metadata\_taxonomy**

The model taxonomy metadata.

**Type**

ModelTaxonomyMetadata

**model\_artifact**

This is built by calling prepare.

**Type**

*ModelArtifact*

**model\_deployment**

A ModelDeployment instance.

**Type**

*ModelDeployment*

**model\_file\_name**

Name of the serialized model.

**Type**

str

**model\_id**

The model ID.

**Type**

str

**properties**

ModelProperties object required to save and deploy model.

**Type**

*ModelProperties*

**runtime\_info**

A RuntimeInfo instance.

**Type**

*RuntimeInfo*

**schema\_input**

Schema describes the structure of the input data.

**Type**

Schema

**schema\_output**

Schema describes the structure of the output data.

**Type**

Schema

**serialize**

Whether to serialize the model to pkl file by default. If False, you need to serialize the model manually, save it under artifact\_dir and update the score.py manually.

**Type**

bool

**version**

The framework version of the model.

**Type**

str

**delete\_deployment(...)**

Deletes the current model deployment.

**deploy(..., *\*\*kwargs*)**

Deploys a model.

**from\_model\_artifact(uri, model\_file\_name, artifact\_dir, ..., *\*\*kwargs*)**

Loads model from the specified folder, or zip/tar archive.

**from\_model\_catalog(model\_id, model\_file\_name, artifact\_dir, ..., *\*\*kwargs*)**

Loads model from model catalog.

**introspect(...)**

Runs model introspection.

**predict(data, ...)**

Returns prediction of input data run against the model deployment endpoint.

**prepare(..., *\*\*kwargs*)**

Prepare and save the score.py, serialized model and runtime.yaml file.

**reload(...)**

Reloads the model artifact files: *score.py* and the *runtime.yaml*.

**save(..., \\*\*kwargs)**  
Saves model artifacts to the model catalog.

**summary\_status(...)**  
Gets a summary table of the current status.

**verify(data, ...)**  
Tests if deployment works in local environment.

## Examples

```
>>> from ads.model.framework.tensorflow_model import TensorFlowModel
>>> import tempfile
>>> import tensorflow as tf
```

```
>>> mnist = tf.keras.datasets.mnist
>>> (x_train, y_train), (x_test, y_test) = mnist.load_data()
>>> x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
>>> tf_estimator = tf.keras.models.Sequential(
...     [
...         tf.keras.layers.Flatten(input_shape=(28, 28)),
...         tf.keras.layers.Dense(128, activation="relu"),
...         tf.keras.layers.Dropout(0.2),
...         tf.keras.layers.Dense(10),
...     ]
... )
>>> loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
>>> tf_estimator.compile(optimizer="adam", loss=loss_fn, metrics=["accuracy"])
>>> tf_estimator.fit(x_train, y_train, epochs=1)
```

```
>>> tf_model = TensorFlowModel(estimator=tf_estimator,
... artifact_dir=tempfile.mkdtemp())
>>> inference_conda_env = "generalml_p37_cpu_v1"
```

```
>>> tf_model.prepare(inference_conda_env="generalml_p37_cpu_v1", force_
↳ overwrite=True)
>>> tf_model.verify(x_test[:1])
>>> tf_model.save()
>>> model_deployment = tf_model.deploy(wait_for_completion=False)
>>> tf_model.predict(x_test[:1])
```

Initiates a TensorFlowModel instance.

### Parameters

- **estimator** (*callable*) – Any model object generated by tensorflow framework
- **artifact\_dir** (*str*) – Directory for generate artifact.
- **properties** ((*ModelProperties*, *optional*). Defaults to *None*.) – ModelProperties object required to save and deploy model.
- **auth** ((*Dict*, *optional*). Defaults to *None*.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys*



or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

#### Returns

TensorFlowModel instance.

#### Return type

*TensorFlowModel*

**get\_data\_serializer**(*data: Union[Dict, str, List, ndarray, Series, DataFrame, tf.Tensor], data\_type: str = None*)

Returns serializable input data.

#### Parameters

- **data** (*Union[Dict, str, list, numpy.ndarray, pd.core.series.Series,]*) –
- **pd.core.frame.DataFrame** – Data expected by the model deployment predict API.
- **tf.Tensor**] – Data expected by the model deployment predict API.
- **data\_type** (*str*) – Type of the data.

#### Returns

A class containing serialized input data and original data type information.

#### Return type

InputDataSerializer

#### Raises

**TypeError** – if provided data type is not supported.

**serialize\_model**(*as\_onnx: bool = False, X\_sample: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, force\_overwrite: bool = False, \*\*kwargs*) → None

Serialize and save Tensorflow model using ONNX or model specific method.

#### Parameters

- **as\_onnx** (*(bool, optional). Defaults to False.*) – If set as True, convert into ONNX model.
- **X\_sample** (*Union[list, tuple, pd.Series, np.ndarray, pd.DataFrame]. Defaults to None.*) – A sample of input data that will be used to generate input schema and detect input\_signature.
- **force\_overwrite** (*(bool, optional). Defaults to False.*) – If set as True, overwrite serialized model if exists.
- **\*\*kwargs** (*optional params used to serialize tensorflow model to onnx,*) –
- **following** (*including the*) – **input\_signature**: a tuple or a list of tf.TensorSpec objects). default to None. Define the shape/dtype of the input so that model(input\_signature) is a valid invocation of the model. **opset\_version**: int. Defaults to None. Used for the ONNX model.

#### Returns

Nothing.

#### Return type

None

**to\_onnx**(*path*: str = None, *input\_signature*=None, *X\_sample*: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, *opset\_version*=None)

Exports the given Tensorflow model into ONNX format.

#### Parameters

- **path** (str, default to None) – Path to save the serialized model.
- **input\_signature** (a tuple or a list of *tf.TensorSpec* objects. default to None.) – Define the shape/dtype of the input so that `model(input_signature)` is a valid invocation of the model.
- **X\_sample** (Union[list, tuple, *pd.Series*, *np.ndarray*, *pd.DataFrame*]. Defaults to None.) – A sample of input data that will be used to generate input schema and detect *input\_signature*.
- **opset\_version** (int. Defaults to None.) – The opset to be used for the ONNX model.

#### Returns

Nothing

#### Return type

None

#### Raises

**ValueError** – if path is not provided

### 18.1.1.16.7 `ads.model.framework.xgboost_model` module

**class** `ads.model.framework.xgboost_model.XGBoostModel`(*estimator*: callable, *artifact\_dir*: str, *properties*: Optional[*ModelProperties*] = None, *auth*: Dict = None, *\*\*kwargs*)

Bases: [\*FrameworkSpecificModel\*](#)

XGBoostModel class for estimators from xgboost framework.

#### **algorithm**

The algorithm of the model.

#### **Type**

str

#### **artifact\_dir**

Artifact directory to store the files needed for deployment.

#### **Type**

str

#### **auth**

Default authentication is set using the `ads.set_auth` API. To override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create an authentication signer to instantiate an *IdentityClient* object.

#### **Type**

Dict

**ds\_client**

The data science client used by model deployment.

**Type**

DataScienceClient

**estimator**

A trained xgboost estimator/model using Xgboost.

**Type**

Callable

**framework**

“xgboost”, the framework name of the model.

**Type**

str

**hyperparameter**

The hyperparameters of the estimator.

**Type**

dict

**metadata\_custom**

The model custom metadata.

**Type**

ModelCustomMetadata

**metadata\_provenance**

The model provenance metadata.

**Type**

ModelProvenanceMetadata

**metadata\_taxonomy**

The model taxonomy metadata.

**Type**

ModelTaxonomyMetadata

**model\_artifact**

This is built by calling prepare.

**Type**

*ModelArtifact*

**model\_deployment**

A ModelDeployment instance.

**Type**

*ModelDeployment*

**model\_file\_name**

Name of the serialized model.

**Type**

str

**model\_id**

The model ID.

**Type**

str

**properties**

ModelProperties object required to save and deploy model.

**Type**

*ModelProperties*

**runtime\_info**

A RuntimeInfo instance.

**Type**

*RuntimeInfo*

**schema\_input**

Schema describes the structure of the input data.

**Type**

Schema

**schema\_output**

Schema describes the structure of the output data.

**Type**

Schema

**serialize**

Whether to serialize the model to pkl file by default. If False, you need to serialize the model manually, save it under artifact\_dir and update the score.py manually.

**Type**

bool

**version**

The framework version of the model.

**Type**

str

**delete\_deployment(...)**

Deletes the current model deployment.

**deploy(..., \\*\*kwargs)**

Deploys a model.

**from\_model\_artifact(uri, model\_file\_name, artifact\_dir, ..., \\*\*kwargs)**

Loads model from the specified folder, or zip/tar archive.

**from\_model\_catalog(model\_id, model\_file\_name, artifact\_dir, ..., \\*\*kwargs)**

Loads model from model catalog.

**introspect(...)**

Runs model introspection.

**predict(data, ...)**

Returns prediction of input data run against the model deployment endpoint.

**prepare**(..., *\\*\*kwargs*)

Prepare and save the `score.py`, serialized model and `runtime.yaml` file.

**reload**(...)

Reloads the model artifact files: `score.py` and the `runtime.yaml`.

**save**(..., *\\*\*kwargs*)

Saves model artifacts to the model catalog.

**summary\_status**(...)

Gets a summary table of the current status.

**verify**(*data*, ...)

Tests if deployment works in local environment.

## Examples

```
>>> import xgboost as xgb
>>> import tempfile
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.datasets import load_iris
>>> from ads.model.framework.xgboost_model import XGBoostModel
```

```
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
>>> xgboost_estimator = xgb.XGBClassifier()
>>> xgboost_estimator.fit(X_train, y_train)
```

```
>>> xgboost_model = XGBoostModel(estimator=xgboost_estimator, artifact_dir=tmp_
↳model_dir)
>>> xgboost_model.prepare(inference_conda_env="generalml_p37_cpu_v1", force_
↳overwrite=True)
>>> xgboost_model.reload()
>>> xgboost_model.verify(X_test)
>>> xgboost_model.save()
>>> model_deployment = xgboost_model.deploy(wait_for_completion=False)
>>> xgboost_model.predict(X_test)
```

Initiates a `XGBoostModel` instance. This class wraps the `XGBoost` model as estimator. It's primary purpose is to hold the trained model and do serialization.

### Parameters

- **estimator** – `XGBoostModel`
- **artifact\_dir** (*str*) – artifact directory to store the files needed for deployment.
- **properties** ((`ModelProperties`, *optional*). Defaults to `None`.) – `ModelProperties` object required to save and deploy model.
- **auth** ((`Dict`, *optional*). Defaults to `None`.) – The default authentication is set using `ads.set_auth` API. If you need to override the default, use the `ads.common.auth.api_keys` or `ads.common.auth.resource_principal` to create appropriate authentication signer and kwargs required to instantiate `IdentityClient` object.

**Returns**

XGBoostModel instance.

**Return type**

*XGBoostModel*

**Examples**

```
>>> import xgboost as xgb
>>> import tempfile
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.datasets import load_iris
>>> from ads.model.framework.xgboost_model import XGBoostModel
```

```
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
```

```
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
>>> train = xgb.DMatrix(X_train, y_train)
>>> test = xgb.DMatrix(X_test, y_test)
>>> xgboost_estimator = XGBClassifier()
>>> xgboost_estimator.fit(X_train, y_train)
>>> xgboost_model = XGBoostModel(estimator=xgboost_estimator, artifact_dir=tempfile.
↳ makedirs())
>>> xgboost_model.prepare(inference_conda_env="generalml_p37_cpu_v1")
>>> xgboost_model.verify(X_test)
>>> xgboost_model.save()
>>> model_deployment = xgboost_model.deploy()
>>> xgboost_model.predict(X_test)
>>> xgboost_model.delete_deployment()
```

**generate\_initial\_types**(*X\_sample: Any*) → List

Auto generate initial types.

**Parameters**

**X\_sample** ((*Any*)) – Train data.

**Returns**

Initial types.

**Return type**

List

**serialize\_model**(*as\_onnx: bool = False, initial\_types: List[Tuple] = None, force\_overwrite: bool = False, X\_sample: Optional[Union[Dict, str, List, Tuple, ndarray, Series, DataFrame]] = None, \*\*kwargs*)

Serialize and save Xgboost model using ONNX or model specific method.

**Parameters**

- **artifact\_dir** (*str*) – Directory for generate artifact.
- **as\_onnx** ((*boolean, optional*)). Defaults to *False*.) – If set as *True*, provide *initial\_types* or *X\_sample* to convert into ONNX.

- **initial\_types** ((*List[Tuple]*, optional). Defaults to None.) – Each element is a tuple of a variable name and a type.
- **force\_overwrite** ((*boolean*, optional). Defaults to False.) – If set as True, overwrite serialized model if exists.
- **X\_sample** (*Union[Dict, str, List, np.ndarray, pd.core.series.Series, pd.core.frame.DataFrame,]*. Defaults to None.) – Contains model inputs such that `model(X_sample)` is a valid invocation of the model. Used to generate `initial_types`.

**Returns**

Nothing.

**Return type**

None

**to\_onnx**(*initial\_types: List[Tuple] = None, X\_sample: Union[list, tuple, DataFrame, Series, ndarray] = None, \*\*kwargs*)

Produces an equivalent ONNX model of the given Xgboost model.

**Parameters**

- **initial\_types** ((*List[Tuple]*, optional). Defaults to None.) – Each element is a tuple of a variable name and a type.
- **X\_sample** (*Union[Dict, str, List, np.ndarray, pd.core.series.Series, pd.core.frame.DataFrame,]*. Defaults to None.) – Contains model inputs such that `model(X_sample)` is a valid invocation of the model. Used to generate `initial_types`.

**Returns**

An ONNX model (type: `ModelProto`) which is equivalent to the input xgboost model.

**Return type**

`onnx.onnx_ml_pb2.ModelProto`

**18.1.1.16.8 Module contents****18.1.1.17 ads.model.runtime package****18.1.1.17.1 Submodules****18.1.1.17.2 ads.model.runtime.env\_info module**

**class** `ads.model.runtime.env_info.EnvInfo`

Bases: `ABC`

Env Info Base class.

**classmethod** `from_path(env_path: str) → EnvInfo`

Initiate an object from a conda pack path.

**Parameters**

**env\_path** (*str*) – conda pack path.

**Returns**

An `EnvInfo` instance.

**Return type**

*EnvInfo*

```
classmethod from_slug(env_slug: str, namespace: str = 'id19sferra6z', bucketname: str = 'service-conda-packs') → EnvInfo
```

Initiate an EnvInfo object from a slug. Only service pack is allowed to use this method.

#### Parameters

- **env\_slug** (*str*) – service pack slug.
- **namespace** (*((str, optional))*) – namespace of region.
- **bucketname** (*((str, optional))*) – bucketname of service pack.

#### Returns

An EnvInfo instance.

#### Return type

*EnvInfo*

```
class ads.model.runtime.env_info.InferenceEnvInfo(inference_env_slug: str = "", inference_env_type: str = "", inference_env_path: str = "", inference_python_version: str = "")
```

Bases: *EnvInfo*, *DataClassSerializable*

Inference conda environment info.

```
inference_env_path: str = ''
```

```
inference_env_slug: str = ''
```

```
inference_env_type: str = ''
```

```
inference_python_version: str = ''
```

```
class ads.model.runtime.env_info.PACK_TYPE(value)
```

Bases: *Enum*

Conda Pack Type

```
SERVICE_PACK = 'data_science'
```

```
USER_CUSTOM_PACK = 'published'
```

```
class ads.model.runtime.env_info.TrainingEnvInfo(training_env_slug: str = "", training_env_type: str = "", training_env_path: str = "", training_python_version: str = "")
```

Bases: *EnvInfo*, *DataClassSerializable*

Training conda environment info.

```
training_env_path: str = ''
```

```
training_env_slug: str = ''
```

```
training_env_type: str = ''
```

```
training_python_version: str = ''
```



18.1.1.17.3 `ads.model.runtime.model_deployment_details` module

```
class ads.model.runtime.model_deployment_details.ModelDeploymentDetails(inference_conda_env:
    ~ads.model.runtime.env_info.InferenceEnvInfo,
    = <factory>)
```

Bases: `DataClassSerializable`

`ModelDeploymentDetails` class.

`inference_conda_env`: [`InferenceEnvInfo`](#)

18.1.1.17.4 `ads.model.runtime.model_provenance_details` module

```
class ads.model.runtime.model_provenance_details.ModelProvenanceDetails(project_ocid: str = "",
    tenancy_ocid: str = "",
    training_code:
    ~ads.model.runtime.model_provenance_details.TrainingCode,
    = <factory>, training_compartment_ocid:
    str = "",
    training_conda_env:
    ~ads.model.runtime.env_info.TrainingEnvInfo,
    = <factory>,
    training_region: str = "",
    training_resource_ocid: str = "",
    user_ocid: str = "",
    vm_image_internal_id: str = "")
```

Bases: `DataClassSerializable`

`ModelProvenanceDetails` class.

`project_ocid`: str = ''

`tenancy_ocid`: str = ''

`training_code`: [`TrainingCode`](#)

`training_compartment_ocid`: str = ''

`training_conda_env`: [`TrainingEnvInfo`](#)

`training_region`: str = ''

`training_resource_ocid`: str = ''

`user_ocid`: str = ''

`vm_image_internal_id`: str = ''

```
class ads.model.runtime.model_provenance_details.TrainingCode(artifact_directory: str = "")
```

Bases: `DataClassSerializable`

`TrainingCode` class.

`artifact_directory`: str = ''

### 18.1.1.17.5 ads.model.runtime.runtime\_info module

```
class ads.model.runtime.runtime_info.RuntimeInfo(model_artifact_version: str = "", model_deployment:
~ads.model.runtime.model_deployment_details.ModelDeploymentDetails, model_provenance:
~ads.model.runtime.model_provenance_details.ModelProvenanceDetails)
    = <factory>, model_provenance:
    = <factory>)
```

Bases: `DataClassSerializable`

`RuntimeInfo` class which is the data class representation of the runtime yaml file.

**classmethod** `from_env()` → *RuntimeInfo*

Populate the `RuntimeInfo` from environment variables.

**Returns**

A `RuntimeInfo` instance.

**Return type**

*RuntimeInfo*

**model\_artifact\_version:** str = ''

**model\_deployment:** *ModelDeploymentDetails*

**model\_provenance:** *ModelProvenanceDetails*

**save()**

Save the `RuntimeInfo` object into runtime.yaml file under the artifact directory.

**Returns**

Nothing.

**Return type**

None

### 18.1.1.17.6 ads.model.runtime.utils module

```
class ads.model.runtime.utils.SchemaValidator(schema_file_path: str)
```

Bases: `object`

Base Schema Validator which validate yaml file.

Initiate a `SchemaValidator` instance.

**Parameters**

**schema\_file\_path** ((str)) – schema file path. The schema is used to validate the yaml file.

**Returns**

A `SchemaValidator` instance.

**Return type**

*SchemaValidator*

**validate**(document: Dict) → bool

Validate the schema.

**Parameters**

**document** ((Dict)) – yaml file content to validate.

**Raises**

**DocumentError** – Raised when the validation schema is missing, has the wrong format or contains errors.:

**Returns**

validation result.

**Return type**

bool

`ads.model.runtime.utils.get_service_packs(namespace: str, bucketname: str) → Tuple[Dict, Dict]`

Get the service pack path mapping and service pack slug mapping. Note: deprecated packs are also included.

**Parameters**

- **namespace** (*str*) – namespace of the service pack.
- **bucketname** (*str*) – bucketname of the service pack.

**Returns**

Service pack path mapping(service pack path -> (slug, python version)) and the service pack slug mapping(service pack slug -> (pack path, python version)).

**Return type**

(Dict, Dict)

**18.1.1.17.7 Module contents****18.1.1.18 ads.oracledb package****18.1.1.18.1 Submodules****18.1.1.18.2 ads.oracledb.oracle\_db module****18.1.1.19 ads.secrets package****18.1.1.19.1 Submodules****18.1.1.19.2 ads.secrets.secrets module**

**class** `ads.secrets.secrets.Secret`

Bases: `object`

Base class

**serialize**(*self*) → `dict`

Serializes attributes as dictionary. Returns dictionary with the keys that are serializable.

**to\_dict**(*self*) → `dict`

returns dictionary with the keys that has *repr* set to True and the value is not None or empty

**export\_dict** -> `dict`

returns dictionary with the keys that has *repr* set to True

**export\_options** -> `dcit`

returns list of attributes with the fields that has *repr* set to True

**export\_dict()** → dict

Serializes attributes as dictionary.

**Returns**

returns dictionary of key/value pair where the value of the attribute is not None and the field does not have *repr*='False'

**Return type**

dict

**export\_options()** → list

Returns list of attributes that have *repr=True*.

**Returns**

returns list of fields that does not have *repr=False*

**Return type**

list

**serialize()** → dict

Serializes attributes as dictionary. An attribute can be marked as not serializable by using *metadata* field of the *field* constructor provided by the *dataclasses* module.

**Returns**

returns dictionary of key/value pair where the value of the attribute is not None and not empty and the field does not have *metadata* = {"serializable":False}. Refer dataclass python documentation for more details about *metadata*

**Return type**

dict

**to\_dict()** → dict

Serializes attributes as dictionary. Returns only non empty attributes.

**Returns**

returns dictionary of key/value pair where the value of the attribute is not None or empty

**Return type**

dict

**class** `ads.secrets.secrets.SecretKeeper`(*content: Optional[bytes] = None, encoded: Optional[str] = None, secret\_id: Optional[str] = None, export\_prefix: str = "", export\_env: bool = False, \*\*kwargs*)

Bases: [Vault](#), `ContextDecorator`

SecretKeeper defines APIs required to serialize and deserialize secrets. Services such as Database, Streaming, and Git require users to provide credentials. These credentials need to be safely accessed at runtime. OCI Vault provides a mechanism for safe storage and access. SecretKeeper uses OCI Vault as a backend to store and retrieve the credentials.

The exact data structure of the credentials varies from service to service.

**Parameters**

- **vault\_id**((*str*, *optional*)). *Default None*) – ocid of the vault
- **key\_id**((*str*, *optional*)). *Default None*) – ocid of the key that is used for encrypting the content
- **compartment\_id** ((*str*, *optional*)). *Default None*) – ocid of the compartment\_id where the vault resides. When available in environment variable - *NB\_SESSION\_COMPARTMENT\_OCID*, will default to that.

- **secret\_client\_auth** ((dict, optional, deprecated since 2.5.1). Default None.) – deprecated since 2.5.1. Use *auth* instead
- **vault\_client\_auth** ((dict, optional, deprecated since 2.5.1). Default None.) – deprecated since 2.5.1. Use *auth* instead
- **auth** ((dict, optional)) – Dictionary returned from `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()`. By default, will follow what is set in *ads.set\_auth*. Use this attribute to override the default.

**decode()** → *SecretKeeper*

Decodes the content in *self.encoded* and sets the value in *self.secret*.

**encode()**

Stores the secret in *self.secret* by calling *serialize* method on *self.data*. Stores base64 encoded string of *self.secret* in *self.encoded*.

**export\_vault\_details**(*filepath: str, format: str = 'json', storage\_options: Optional[dict] = None*)

Save *secret\_id* in a json file

#### Parameters

- **filepath** (*str*) – Filepath to save the file.
- **format** (*str*) – Default is *json*. Valid values:
  - *yaml* or *yml* - to store vault details in a yaml file
  - *json* - to store vault details in a json file
- **storage\_options** (*dict, optional.*) – *storage\_options* dict as required by *fsspec* library

#### Returns

Returns None

#### Return type

None

**classmethod load\_secret**(*source: str, format: str = 'ocid', export\_env: bool = False, export\_prefix: str = "", auth=None, storage\_options: Optional[dict] = None, \*\*kwargs*) → Union[dict, *SecretKeeper*]

Loads secret from vault using *secret\_id*.

#### Parameters

- **source** (*str*) – Source could be one of the following:
  - OCID of the secret that has the secret content.
  - file path that is json or yaml format with the key - *secret\_id: ocid1.vaultsecret.<unique\_ID>*
- **format** (*str*) – Default is *ocid*. When *ocid*, the source must be a secret id Value values:
  - *ocid* - source is expected to be ocid of the secret
  - *yaml* or *yml* - source is expected to be a path to a valid yaml file
  - *json* - source is expected to be a path to a valid json file
- **export\_env** (*str, Default False*) – When set to true, the credentials will be exported to the environment variable. When *load\_secret* is invoked using *with* statement, information exported as environment variable is unset before leaving the *with* scope

- **export\_prefix** (*str*, *Default ""*) – Prefix to the environment variable that is exported.
- **auth** (*dict*, *optional*) – By default authentication will follow what is configured using `ads.set_auth` API. Accepts dict returned from `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()`.
- **storage\_options** (*dict*, *optional*) – `storage_options` dict as required by `fsspec` library
- **kwargs** – key word arguments accepted by the constructor of the class from which this method is invoked.

#### Returns

- *dict* – When called from within *with* block, Returns a dictionary containing the secret
- `ads.secrets.SecretKeeper` – When called without using *with* operator.

#### Examples

```
>>> from ads.secrets import APIKeySecretKeeper
>>> with APIKeySecretKeeper.load_secret(source="ocid1.vaultsecret.**<unique_ID>
→**",
...                                     export_prefix="mykafka",
...                                     export_env=True
...                                     ) as apisecret:
...     import os
...     print("Credentials inside environment variable:",
...           os.environ.get('mykafka.api_key'))
...     print("Credentials inside `apisecret` object: ", apisecret)
Credentials inside environment variable: <your api key>
Credentials inside `apisecret` object: {'api_key': 'your api key'}
```

```
>>> from ads.secrets import ADBSecretKeeper
>>> with ADBSecretKeeper.load_secret("ocid1.vaultsecret.**<unique_ID>**") as_
→adw_creds2:
...     import pandas as pd
...     df2 = pd.DataFrame(ads.read_sql("select * from ATTRITION_DATA",
...                                     connection_parameters=adw_creds2)
...     print(df2.head(2))
          JOBFUNCTION ATTRITION
0  Product Management      No
1  Software Developer      No
```

```
required_keys = ['secret_id']
```

```
save(name: str, description: str, freeform_tags: Optional[dict] = None, defined_tags: Optional[dict] =
None) → SecretKeeper
```

Saves credentials to Vault and returns self.

#### Parameters

- **name** (*str*) – Name of the secret when saved in the Vault.
- **description** (*str*) – Description of the secret when saved in the Vault.

- **freeform\_tags** (*dict*, *optional*) – freeform\_tags to be used for saving the secret in OCI console.
- **defined\_tags** (*dict*, *optional*.) – Save the tags under predefined tags in OCI console.

**Returns**

Returns self object.

**Return type**

*SecretKeeper*

**to\_dict()** → dict

Returns dict of credentials retrieved from the vault or set through constructor arguments.

**Returns**

dict of credentials retrieved from the vault or set through constructor.

**Return type**

dict

**18.1.1.19.3 ads.secrets.adb module**

```
class ads.secrets.adb.ADBSecret(user_name: str, password: str, service_name: str, wallet_location:
                               ~typing.Optional[str] = None, wallet_file_name: ~typing.Optional[str] =
                               None, wallet_content: ~typing.Optional[dict] = None, wallet_secret_ids:
                               list = <factory>)
```

Bases: *Secret*

Dataclass representing the attributes managed and serialized by ADBSecretKeeper

**password:** str

**service\_name:** str

**user\_name:** str

**wallet\_content:** dict = None

**wallet\_file\_name:** str = None

**wallet\_location:** str = None

**wallet\_secret\_ids:** list

```
class ads.secrets.adb.ADBSecretKeeper(user_name: Optional[str] = None, password: Optional[str] =
                                       None, service_name: Optional[str] = None, wallet_location:
                                       Optional[str] = None, wallet_dir: Optional[str] = None,
                                       repository_path: Optional[str] = None, repository_key:
                                       Optional[str] = None, **kwargs)
```

Bases: *SecretKeeper*

*ADBSecretKeeper* provides an interface to save ADW/ATP database credentials. This interface does not store the wallet file by default. For saving wallet file, set *save\_wallet=True* while calling *ADBSecretKeeper.save* method.

## Examples

```
>>> # Saving credentials without saving the wallet file
>>> from ads.secrets.adw import ADBSecretKeeper
>>> vault_id = "ocidl.vault.oc1..<unique_ID>"
>>> key_id = "ocidl.key..<unique_ID>"
```

```
>>> import ads
>>> ads.set_auth("resource_principal") # If using resource principal for
↳ authentication
>>> connection_parameters={
...     "user_name":"admin",
...     "password":"<your password>",
...     "service_name":"service_name_{high|low|med}",
...     "wallet_location":"/home/datascience/Wallet_xxxx.zip"
... }
>>> adw_keeper = ADBSecretKeeper(vault_id=vault_id, key_id=key_id, **connection_
↳ parameters)
>>> adw_keeper.save("adw_employee", "My DB credentials", freeform_tags={"schema":
↳ "emp"}) # Does not save the wallet file
>>> print(adw_keeper.secret_id) # Prints the secret_id of the stored credentials
>>> adw_keeper.export_vault_details("adw_employee_att.json", format="json") # Save
↳ the secret id and vault info to a json file
```

```
>>> # Loading credentials
>>> import ads
>>> ads.set_auth("resource_principal") # If using resource principal for
↳ authentication
>>> from ads.secrets.adw import ADBSecretKeeper
>>> secret_id = "ocidl.vaultsecret.oc1..<unique_ID>"
>>> with ADBSecretKeeper.load_secret(source=secret_id,
                                   wallet_location='/home/datascience/Wallet_xxxxxx.zip')
↳ as adw_creds:
...     import pandas as pd
...     df = pd.DataFrame(ads.read_sql("select * from EMPLOYEE", connection_
↳ parameters=adw_creds))
```

```
>>> myadw_creds = ADBSecretKeeper.load_secret(source='adw_employee_att.json',
↳ format="json"
...     wallet_location='/home/datascience/Wallet_xxxxxx.zip')
>>> pd.DataFrame(ads.read_sql("select * from ATTRITION_DATA", connection_
↳ parameters=myadw_creds.to_dict()).head(2))
```

```
>>> # Saving and loading credentials with wallet storage
>>> # Saving credentials
>>> from ads.secrets.adw import ADBSecretKeeper
>>> vault_id = "ocidl.vault.oc1..<unique_ID>"
>>> key_id = "ocidl.key.oc1..<unique_ID>"
```

```
>>> import ads
>>> ads.set_auth("resource_principal") # If using resource principal for
↳ authentication
```

(continues on next page)



(continued from previous page)

```
>>> connection_parameters={
...     "user_name":"admin",
...     "password":"<your password>",
...     "service_name":"service_name_{high|low|med}",
...     "wallet_location":"/home/datascience/Wallet_xxxx.zip"
... }
>>> adw_keeper = ADBSecretKeeper(vault_id=vault_id, key_id=key_id, **connection_
↳parameters)
>>> adw_keeper.save("adw_employee", "My DB credentials", freeform_tags={"schema":
↳"emp"}, save_wallet=True)
>>> print(adw_keeper.secret_id) # Prints the secret_id of the stored credentials
>>> adw_keeper.export_vault_details("adw_employee_att.json") # Save the secret id,
↳and vault info to a json file
```

```
>>> # Loading credentials
>>> import ads
>>> ads.set_auth("resource_principal") # If using resource principal for
↳authentication
>>> from ads.secrets.adw import ADBSecretKeeper
>>> secret_id = "ocidl.vaultsecret.oc1..<unique_ID>"
>>> with ADBSecretKeeper.load_secret(source=secret_id) as adw_creds:
...     import pandas as pd
...     df = pd.DataFrame(ads.read_sql("select * from EMPLOYEE", connection_
↳parameters=adw_creds))
```

```
>>> myadw_creds = ADBSecretKeeper.load_secret(source='adw_employee_att.json',
↳format='json')
>>> pd.DataFrame(ads.read_sql("select * from ATTRITION_DATA", connection_
↳parameters=myadw_creds.to_dict()).head(2))
```

### Parameters

- **user\_name** ((str, optional). Default None) – user\_name of the database
- **password** ((str, optional). Default None) – password for connecting to the database
- **service\_name** ((str, optional). Default None) – service name of the ADB instance
- **wallet\_location** ((str, optional). Default None) – full path to the wallet zip file used for connecting to ADB instance.
- **wallet\_dir** ((str, optional). Default None) – local directory where the extracted wallet content is saved
- **repository\_path** ((str, optional). Default None.) – Path to credentials repository. For more details refer *ads.database.connection*
- **repository\_key** ((str, optional). Default None.) – Configuration key for loading the right configuration from repository. For more details refer *ads.database.connection*
- **kwargs** – vault\_id: str. OCID of the vault where the secret is stored. Required for saving secret. key\_id: str. OCID of the key used for encrypting the secret. Required for saving secret. compartment\_id: str. OCID of the compartment where the vault is located. Required for saving secret. auth: dict. Dictionary returned from *ads.common.auth.api\_keys()* or

`ads.common.auth.resource_principal()`. By default, will follow what is set in `ads.set_auth`. Use this attribute to override the default.

**decode()** → *ADBSecretKeeper*

Converts the content in `self.secret` to *ADBSecret* and stores in `self.data`

If the `wallet_location` is passed through the constructor, then retain it. We do not want to override what user has passed in. If the `wallet_location` was not passed, but the secret has `wallet_secret_ids`, then we generate the wallet zip file in the location specified by `wallet_dir` in the constructor

**Returns**

Returns self object

**Return type**

*ADBSecretKeeper*

**encode**(*serialize\_wallet: bool = False*) → *ADBSecretKeeper*

Prepares content to save in vault. The `user_name`, `password` and `service_name` and the individual files inside the wallet zip file are base64 encoded and stored in `self.secret`

**Parameters**

**serialize\_wallet** (*bool, optional*) – When set to True, loads the wallet zip file and encodes the content of each file in the zip file.

**Returns**

Returns self object

**Return type**

*ADBSecretKeeper*

**save**(*name: str, description: str, freeform\_tags: Optional[dict] = None, defined\_tags: Optional[dict] = None, save\_wallet: bool = False*) → *ADBSecretKeeper*

Saves credentials to Vault and returns self.

**Parameters**

- **name** (*str*) – Name of the secret when saved in the Vault.
- **description** (*str*) – Description of the secret when saved in the Vault.
- **freeform\_tags** (*(dict, optional). Default is None*) – freeform\_tags to be used for saving the secret in OCI console.
- **defined\_tags** (*(dict, optional). Default is None*) – Save the tags under pre-defined tags in OCI console.
- **save\_wallet** (*(bool, optional). Default is False*) – If set to True, saves the contents of the wallet file as separate secret.

**Returns**

Returns self object

**Return type**

*ADBSecretKeeper*

### 18.1.1.19.4 ads.secrets.mysqladb module

```
class ads.secrets.mysqladb.MySQLDBSecret(user_name: str, password: str, host: str, port: str, database:  
Optional[str] = None)
```

Bases: *Secret*

Dataclass representing the attributes managed and serialized by MySQLDBSecretKeeper

**database:** **str** = **None**

**host:** **str**

**password:** **str**

**port:** **str**

**user\_name:** **str**

```
class ads.secrets.mysqladb.MySQLDBSecretKeeper(user_name: Optional[str] = None, password:  
Optional[str] = None, host: Optional[str] = None, port:  
str = '3306', database: Optional[str] = None,  
repository_path: Optional[str] = None, repository_key:  
Optional[str] = None, **kwargs)
```

Bases: *SecretKeeper*

*MySQLDBSecretKeeper* provides an interface to save MySQL database credentials. If you use Wallet file for connecting to the database, please use *ADBSecretKeeper*.

### Examples

```
>>> from ads.secrets.mysqladb import MySQLDBSecretKeeper
>>> vault_id = "ocidl.vault.oc1.<unique_ID>"
>>> key_id = "ocidl.key.<unique_ID>"
```

```
>>> import ads
>>> ads.set_auth("resource_principal") # If using resource principal for
↳ authentication
>>> connection_parameters={
...     "user_name": "<your user name>",
...     "password": "<your password>",
...     "host": "<db host>",
...     "port": "<db port>",
...     "database": "<database>",
... }
>>> mysqladb_keeper = MySQLDBSecretKeeper(vault_id=vault_id, key_id=key_id,
↳ **connection_parameters)
>>> mysqladb_keeper.save("mysqladb_employee", "My DB credentials", freeform_tags={
↳ "schema": "emp"})
>>> print(mysqladb_keeper.secret_id) # Prints the secret_id of the stored credentials
>>> mysqladb_keeper.export_vault_details("mysqladb_employee_att.json") # Save the
↳ secret id and vault info to a json file
```

```

>>> # Loading credentials
>>> import ads
>>> ads.set_auth("resource_principal") # If using resource principal for
↪ authentication
>>> from ads.secrets.mysqladb import MySQLDBSecretKeeper
>>> secret_id = "ocidl.vaultsecret.oc1.<unique_ID>"
>>> with MySQLDBSecretKeeper.load_secret(source=secret_id) as mysqladb_creds:
...     import pandas as pd
...     df = pd.DataFrame(ads.read_sql("select * from EMPLOYEE", connection_
↪ parameters=mysqladb_creds, engine="mysql"))

>>> mymysqladb_creds = MySQLDBSecretKeeper.load_secret(source='mysqladb_employee_att.
↪ json', format="json")
>>> pd.DataFrame(ads.read_sql("select * from ATTRITION_DATA", connection_
↪ parameters=mymysqladb_creds.to_dict(), engine="mysql")).head(2)

```

### Parameters

- **user\_name** ((*str*, optional). Default None) – user\_name of the database
- **password** ((*str*, optional). Default None) – password for connecting to the database
- **host** ((*str*, optional). Default None) – Database host name
- **port** ((*str*, optional). Default 1521) – Port number
- **database** ((*str*, optional). Default None) – database name
- **repository\_path** ((*str*, optional). Default None.) – Path to credentials repository. For more details refer *ads.database.connection*
- **repository\_key** ((*str*, optional). Default None.) – Configuration key for loading the right configuration from repository. For more details refer *ads.database.connection*
- **kwargs** – vault\_id: str. OCID of the vault where the secret is stored. Required for saving secret. key\_id: str. OCID of the key used for encrypting the secret. Required for saving secret. compartment\_id: str. OCID of the compartment where the vault is located. Required for saving secret. auth: dict. Dictionary returned from *ads.common.auth.api\_keys()* or *ads.common.auth.resource\_principal()*. By default, will follow what is set in *ads.set\_auth*. Use this attribute to override the default.

**decode()** → *MySQLDBSecretKeeper*

Converts the content in *self.encoded* to *MySQLDBSecret* and stores in *self.data*

### Returns

Returns self object

### Return type

*MySQLDBSecretKeeper*

## 18.1.1.19.5 ads.secrets.oracledb module

```
class ads.secrets.oracledb.OracleDBSecret(user_name: str, password: str, host: str, port: str,
                                          service_name: Optional[str] = None, sid: Optional[str] =
                                          None, dsn: Optional[str] = None)
```

Bases: [Secret](#)

Dataclass representing the attributes managed and serialized by OracleDBSecretKeeper

**dsn: str = None**

**host: str**

**password: str**

**port: str**

**service\_name: str = None**

**sid: str = None**

**user\_name: str**

```
class ads.secrets.oracledb.OracleDBSecretKeeper(user_name: Optional[str] = None, password:
                                                Optional[str] = None, service_name: Optional[str] =
                                                None, sid: Optional[str] = None, host: Optional[str]
                                                = None, port: str = '1521', dsn: Optional[str] =
                                                None, repository_path: Optional[str] = None,
                                                repository_key: Optional[str] = None, **kwargs)
```

Bases: [SecretKeeper](#)

*OracleDBSecretKeeper* provides an interface to save Oracle database credentials. If you use Wallet file for connecting to the database, please use *ADBSecretKeeper*.

## Examples

```
>>> from ads.secrets.oracledb import OracleDBSecretKeeper
>>> vault_id = "ocidl.vault.oc1.<unique_ID>"
>>> key_id = "ocidl.key.<unique_ID>"
```

```
>>> import ads
>>> ads.set_auth("resource_principal") # If using resource principal for
↪ authentication
>>> connection_parameters={
...     "user_name": "<your user name>",
...     "password": "<your password>",
...     "service_name": "service_name",
...     "host": "<db host>",
...     "port": "<db port>",
... }
>>> oracledb_keeper = OracleDBSecretKeeper(vault_id=vault_id, key_id=key_id,
↪ **connection_parameters)
>>> oracledb_keeper.save("oracledb_employee", "My DB credentials", freeform_tags={
↪ "schema": "emp"})
```

(continues on next page)

(continued from previous page)

```
>>> print(oracledb_keeper.secret_id) # Prints the secret_id of the stored_
↳ credentials
>>> oracledb_keeper.export_vault_details("oracledb_employee_att.json") # Save the_
↳ secret id and vault info to a json file
```

```
>>> # Loading credentials
>>> import ads
>>> ads.set_auth("resource_principal") # If using resource principal for_
↳ authentication
>>> from ads.secrets.oracledb import OracleDBSecretKeeper
>>> secret_id = "ocidl.vaultsecret.ocl.<unique_ID>"
>>> with OracleDBSecretKeeper.load_secret(source=secret_id) as oracledb_creds:
...     import pandas as pd
...     df = pd.DataFrame(ads.read_sql("select * from EMPLOYEE", connection_
↳ parameters=oracledb_creds)
```

```
>>> myoracledb_creds = OracleDBSecretKeeper.load_secret(source='oracledb_employee_
↳ att.json', format="json")
>>> pd.DataFrame(ads.read_sql("select * from ATTRITION_DATA", connection_
↳ parameters=myoracledb_creds.to_dict()).head(2)
```

### Parameters

- **user\_name** ((str, optional). Default None) – user\_name of the database
- **password** ((str, optional). Default None) – password for connecting to the database
- **service\_name** ((str, optional). Default None) – service name of the Oracle DB instance
- **sid** ((str, optional). Default None) – Provide sid if service name is not available.
- **host** ((str, optional). Default None) – Database host name
- **port** ((str, optional). Default 1521) – Port number
- **dsn** ((str, optional). Default None) – dsn string for connecting with oracledb. Refer *cx\_Oracle* documentation
- **repository\_path** ((str, optional). Default None.) – Path to credentials repository. For more details refer *ads.database.connection*
- **repository\_key** ((str, optional). Default None.) – Configuration key for loading the right configuration from repository. For more details refer *ads.database.connection*
- **kwargs** – vault\_id: str. OCID of the vault where the secret is stored. Required for saving secret. key\_id: str. OCID of the key used for encrypting the secret. Required for saving secret. compartment\_id: str. OCID of the compartment where the vault is located. Required for saving secret. auth: dict. Dictionary returned from *ads.common.auth.api\_keys()* or *ads.common.auth.resource\_principal()*. By default, will follow what is set in *ads.set\_auth*. Use this attribute to override the default.

**decode()** → *OracleDBSecretKeeper*

Converts the content in *self.encoded* to *OracleDBSecret* and stores in *self.data*

**Returns**

Returns self object

**Return type***OracleDBSecretKeeper***18.1.1.19.6 ads.secrets.big\_data\_service module**

```
class ads.secrets.big_data_service.BDSSecret(principal: str, hdfs_host: str, hive_host: str, hdfs_port:
    str, hive_port: str, kerb5_path: ~typing.Optional[str] =
    None, kerb5_content: ~typing.Optional[dict] = None,
    keytab_path: ~typing.Optional[str] = None,
    keytab_content: ~typing.Optional[dict] = None,
    secret_id: str = <factory>)
```

Bases: *Secret*

Dataclass representing the attributes managed and serialized by BDSSecretKeeper.

**principal**

The unique identity to which Kerberos can assign tickets.

**Type**

str

**hdfs\_host**

hdfs host name from the bds cluster.

**Type**

str

**hive\_host**

hive host name from the bds cluster.

**Type**

str

**hdfs\_port**

hdfs port from the bds cluster.

**Type**

str

**hive\_port**

hive port from the bds cluster.

**Type**

str

**kerb5\_path**

krb5.conf file path.

**Type**

str

**kerb5\_content**

Content of the krb5.conf.

**Type**

dict

**keytab\_path**

Path to the keytab file.

**Type**

str

**keytab\_content**

Content of the keytab file.

**Type**

dict

**secret\_id**

secret id where the BDSSecret is stored.

**Type**

str

**hdfs\_host: str**

**hdfs\_port: str**

**hive\_host: str**

**hive\_port: str**

**kerb5\_content: dict = None**

**kerb5\_path: str = None**

**keytab\_content: dict = None**

**keytab\_path: str = None**

**principal: str**

**secret\_id: str**

```
class ads.secrets.big_data_service.BDSSecretKeeper(principal: Optional[str] = None, hdfs_host:
Optional[str] = None, hive_host: Optional[str] =
None, hdfs_port: Optional[str] = None,
hive_port: Optional[str] = None, kerb5_path:
Optional[str] = None, kerb5_content:
Optional[str] = None, keytab_path: Optional[str]
= None, keytab_content: Optional[str] = None,
keytab_dir: Optional[str] = None, secret_id:
Optional[str] = None, **kwargs)
```

Bases: [SecretKeeper](#)

*BDSSecretKeeper* provides an interface to save BDS hdfs and hive credentials. This interface does not store the wallet file by default. For saving keytab and krb5.conf file, set *save\_files=True* while calling *BDSSecretKeeper.save* method.

**principal**

The unique identity to which Kerberos can assign tickets.

**Type**

str



**hdfs\_host**

hdfs host name from the bds cluster.

**Type**

str

**hive\_host**

hive host name from the bds cluster.

**Type**

str

**hdfs\_port**

hdfs port from the bds cluster.

**Type**

str

**hive\_port**

hive port from the bds cluster.

**Type**

str

**kerb5\_path**

krb5.conf file path.

**Type**

str

**kerb5\_content**

Content of the krb5.conf.

**Type**

dict

**keytab\_path**

Path to the keytab file.

**Type**

str

**keytab\_content**

Content of the keytab file.

**Type**

dict

**secret\_id**

secret id where the BDSSecret is stored.

**Type**

str

**kwargs**

-----

**vault\_id**

**Type**

str. OCID of the vault where the secret is stored. Required for saving secret.

**key\_id**

**Type**

str. OCID of the key used for encrypting the secret. Required for saving secret.

**compartment\_id**

**Type**

str. OCID of the compartment where the vault is located. Required for saving secret.

**auth**

**Type**

dict. Dictionary returned from `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()`. By default, will follow what is set in `ads.set_auth`. Use this attribute to override the default.

#### Parameters

- **principal** (*str*) – The unique identity to which Kerberos can assign tickets.
- **hdfs\_host** (*str*) – hdfs host name from the bds cluster.
- **hive\_host** (*str*) – hive host name from the bds cluster.
- **hdfs\_port** (*str*) – hdfs port from the bds cluster.
- **hive\_port** (*str*) – hive port from the bds cluster.
- **kerb5\_path** (*str*) – krb5.conf file path.
- **kerb5\_content** (*dict*) – Content of the krb5.conf.
- **keytab\_path** (*str*) – Path to the keytab file.
- **keytab\_content** (*dict*) – Content of the keytab file.
- **keytab\_dir** (*((str, optional).)*) – Default None. Local directory where the extracted keytab content is saved.
- **secret\_id** (*str*) – secret id where the BDSSecret is stored.

#### kwargs

**vault\_id**: str. OCID of the vault where the secret is stored. Required for saving secret. **key\_id**: str. OCID of the key used for encrypting the secret. Required for saving secret. **compartment\_id**: str. OCID of the compartment where the vault is located. Required for saving secret. **auth**: dict. Dictionary returned from `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()`. By default, will follow what is set in `ads.set_auth`. Use this attribute to override the default.

**decode**(*save\_files: bool = True*) → `ads.secrets.bds.BDSSecretKeeper`

Converts the content in `self.secret` to *BDSSecret* and stores in `self.data`

If the *keytab\_path* and *kerb5\_path* are passed through the constructor, then retain it. We do not want to override what user has passed in. If the *keytab\_path* and *kerb5\_path* are not passed, but the secret has *secret\_id*, then we generate the keytab file in the location specified by *keytab\_path* in the constructor.

**Returns**

Returns self object

**Return type**

*BDSSecretKeeper*

**encode**(*serialize: bool = True*) → ads.secrets.bds.BDSSecretKeeper

Prepares content to save in vault. The port, host name and the keytab and krb5.config files are base64 encoded and stored in *self.secret*

**Parameters**

**serialize** (*bool, optional*) – When set to True, loads the keytab and krb5.config file and encodes the content of both files.

**Returns**

Returns self object

**Return type**

*BDSSecretKeeper*

**save**(*name: str, description: str, freeform\_tags: dict = None, defined\_tags: dict = None, save\_files: bool = True*) → ads.secrets.bds.BDSSecretKeeper

Saves credentials to Vault and returns self.

**Parameters**

- **name** (*str*) – Name of the secret when saved in the Vault.
- **description** (*str*) – Description of the secret when saved in the Vault.
- **freeform\_tags** ((*dict, optional*). *Default is None*) – freeform\_tags to be used for saving the secret in OCI console.
- **defined\_tags** ((*dict, optional*). *Default is None*) – Save the tags under pre-defined tags in OCI console.
- **save\_files** ((*bool, optional*). *Default is False*) – If set to True, saves the contents of the keytab and krb5 file as separate secret.

**Returns**

Returns self object

**Return type**

*BDSSecretKeeper*

**18.1.1.19.7 ads.secrets.auth\_token module**

**class** ads.secrets.auth\_token.AuthToken(*auth\_token: str*)

Bases: *Secret*

AuthToken dataclass holds *auth\_token* attribute

**auth\_token: str**

**class** ads.secrets.auth\_token.AuthTokenSecretKeeper(*auth\_token=None, \*\*kwargs*)

Bases: *SecretKeeper*

*AuthTokenSecretKeeper* uses *ads.secrets.auth\_token.AuthToken* class to manage Auth Token credentials. The credentials are stored in Vault as a dictionary with the following format - {"auth\_token": "user provided value"}

## Examples

```
>>> from ads.secrets.auth_token import AuthTokenSecretKeeper
>>> import ads
>>> ads.set_auth("resource_principal") #If using resource principal for
↪ authentication
>>> # Save Auth Tokens or Access Keys to the vault
>>>
>>> authtoken2 = AuthTokenSecretKeeper(vault_id=vault_id,
...     key_id=key_id,
...     auth_token="<your auth token>").save("my_xyz_auth_token2",
...     "This is my
↪ auth token for git repo xyz",
...     freeform_tags={
↪ "gitrepo": "xyz"})
>>> authtoken2.export_vault_details("my_git_token_vault_info.yaml", format="yaml")
>>> # Loading credentials
>>> with AuthTokenSecretKeeper.load_secret(source="ocid1.vaultsecret.oc1..<unique_
↪ ID>",
...     export_prefix="mygitrepo",
...     export_env=True
... ) as authtoken:
...     import os
...     print("Credentials inside environment variable:", os.environ.get('mygitrepo.
↪ auth_token'))
...     print("Credentials inside `authtoken` object: ", authtoken)
Credentials inside environment variable: <your auth token>
Credentials inside `authtoken` object: {'auth_token': '<your auth token>'}
>>> print("Credentials inside `authtoken` object: ", authtoken)
Credentials inside `authtoken` object: {'auth_token': None}
>>> print("Credentials inside environment variable:", os.environ.get('mygitrepo.
↪ auth_token'))
Credentials inside environment variable: None
```

## Parameters

- **auth\_token** ((*str*, optional). Default *None*) – auth token string that needs to be stored in the vault
- **kwargs** – *vault\_id*: str. OCID of the vault where the secret is stored. Required for saving secret. *key\_id*: str. OCID of the key used for encrypting the secret. Required for saving secret. *compartment\_id*: str. OCID of the compartment where the vault is located. Required for saving secret. *auth*: dict. Dictionary returned from `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()`. By default, will follow what is set in `ads.set_auth`. Use this attribute to override the default.

`decode()` → *AuthTokenSecretKeeper*

Converts the content in *self.encoded* to *AuthToken* and stores in *self.data*

## Returns

Returns the self object after decoding *self.encoded* and updates *self.data*

## Return type

*AuthTokenSecretKeeper*

### 18.1.1.19.8 Module contents

### 18.1.1.20 ads.text\_dataset package

#### 18.1.1.20.1 Submodules

#### 18.1.1.20.2 ads.text\_dataset.backends module

**class** ads.text\_dataset.backends.Base

Bases: object

Base class for backends.

**convert\_to\_text**(*fhandler: OpenFile, dst\_path: str, fname: Optional[str] = None, storage\_options: Optional[Dict] = None*) → str

Convert input file to a text file

**Parameters**

- **fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*
- **dst\_path** (*str*) – local folder or cloud storage prefix to save converted text files
- **fname** (*str, optional*) – filename for converted output, relative to dirname or prefix, by default None
- **storage\_options** (*dict, optional*) – storage options for cloud storage

**Returns**

path to saved output

**Return type**

str

**get\_metadata**(*fhandler: OpenFile*) → Dict

Get metadata of a file.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Returns**

dictionary of metadata

**Return type**

dict

**read\_line**(*fhandler: OpenFile*) → Generator[Union[str, List[str]], None, None]

Read lines from a file.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Yields**

*Generator* – a generator that yields lines

**read\_text**(*fhandler: OpenFile*) → Generator[Union[str, List[str]], None, None]

Read entire file into a string.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Yields**

*Generator* – a generator that yields text in the file

**class** `ads.text_dataset.backends.PDFPlumber`

Bases: [\*Base\*](#)

**convert\_to\_text**(*fhandler*, *dst\_path*, *fname=None*, *storage\_options=None*)

Convert input file to a text file

**Parameters**

- **fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*
- **dst\_path** (*str*) – local folder or cloud storage prefix to save converted text files
- **fname** (*str*, *optional*) – filename for converted output, relative to dirname or prefix, by default *None*
- **storage\_options** (*dict*, *optional*) – storage options for cloud storage

**Returns**

path to saved output

**Return type**

*str*

**get\_metadata**(*fhandler*)

Get metadata of a file.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Returns**

dictionary of metadata

**Return type**

*dict*

**read\_line**(*fhandler*)

Read lines from a file.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Yields**

*Generator* – a generator that yields lines

**read\_text**(*fhandler*)

Read entire file into a string.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Yields**

*Generator* – a generator that yields text in the file

**class** `ads.text_dataset.backends.Tika`

Bases: [\*Base\*](#)

**convert\_to\_text**(*fhandler*, *dst\_path*, *fname=None*, *storage\_options=None*)

Convert input file to a text file

**Parameters**

- **fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*
- **dst\_path** (*str*) – local folder or cloud storage prefix to save converted text files
- **fname** (*str, optional*) – filename for converted output, relative to dirname or prefix, by default None
- **storage\_options** (*dict, optional*) – storage options for cloud storage

**Returns**

path to saved output

**Return type**

str

**detect\_encoding**(*fhandler: OpenFile*)

**get\_metadata**(*fhandler*)

Get metadata of a file.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Returns**

dictionary of metadata

**Return type**

dict

**read\_line**(*fhandler*)

Read lines from a file.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Yields**

*Generator* – a generator that yields lines

**read\_text**(*fhandler*)

Read entire file into a string.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Yields**

*Generator* – a generator that yields text in the file

### 18.1.1.20.3 ads.text\_dataset.dataset module

**class** ads.text\_dataset.dataset.**DataLoader**(*engine: Optional[str] = None*)

Bases: object

DataLoader binds engine, FileProcessor and File handler(in this case it is *fsspec*) together to produce a dataframe of parsed text from files.

This class is expected to be used mainly from TextDatasetFactory class.

**processor**

processor that is used for loading data.

**Type**

*ads.text\_dataset.extractor.FileProcessor*

## Examples

```
>>> import oci
>>> from ads.text_dataset.dataset import TextDatasetFactory as textfactory
>>> from ads.text_dataset.options import Options
>>> df = textfactory.format('pdf').engine('pandas').read_line(
...     'oci://<bucket-name>@<namespace>/<path>/*.pdf',
...     storage_options={"config": oci.config.from_file(os.path.join("~/oci",
↪ "config"))}),
... )
>>> data_gen = textfactory.format('pdf').option(Options.FILE_NAME).backend(
↪ 'pdfplumber').read_text(
...     'oci://<bucket-name>@<namespace>/<path>/*.pdf',
...     storage_options={"config": oci.config.from_file(os.path.join("~/oci",
↪ "config"))}),
... )
>>> textfactory.format('docx').convert_to_text(
...     'oci://<bucket-name>@<namespace>/<path>/*.docx',
...     './extracted',
...     storage_options={"config": oci.config.from_file(os.path.join("~/oci",
↪ "config"))}),
... )
>>> textfactory.format('docx').convert_to_text(
...     'oci://<bucket-name>@<namespace>/<path>/*.docx',
...     'oci://<bucket-name>@<namespace>/<out_path>',
...     storage_options={"config": oci.config.from_file(os.path.join("~/oci",
↪ "config"))}),
... )
>>> meta_gen = textfactory.format('docx').metadata_schema(
...     'oci://<bucket-name>@<namespace>/papers/*.pdf',
...     storage_options={"config": oci.config.from_file(os.path.join("~/oci",
↪ "config"))}),
... )
>>> df = textfactory.format('pdf').engine('pandas').option(Options.FILE_METADATA, {
↪ 'extract': ['Author']}).read_text(
...     'oci://<bucket-name>@<namespace>/<path>/*.pdf',
...     storage_options={"config": oci.config.from_file(os.path.join("~/oci",
↪ "config"))}),
...     total_files=10,
... )
>>> df = textfactory.format('txt').engine('cudf').read_line(
...     'oci://<bucket-name>@<namespace>/<path>/*.log',
...     udf=r'^[(\S+)\s(\S+)\s(\d+)\s(\d+:\d+:\d+)\s(\d+)]\s(\S+)\s(\S+)\s(\S+)\s(\S+)',
↪ s(\S+)',
...     df_args={"columns": ["day", "month", "date", "time", "year", "type", "method",
↪ "status", "file"]},
...     n_lines_per_file=10,
... )
```

Initialize a DataLoader object.

### Parameters

**engine** (*str*, *optional*) – dataframe engine, by default None.

### Return type



None

**backend**(*backend*: Union[str, Base]) → None

Set backend used for extracting text from files.

**Parameters**

**backend** ((str | *ads.text\_dataset.backends.Base*)) – backend for extracting text from raw files.

**Return type**

None

**convert\_to\_text**(*src\_path*: str, *dst\_path*: str, *encoding*: str = 'utf-8', *storage\_options*: Optional[Dict] = None) → None

Convert files to plain text files.

**Parameters**

- **src\_path** (str) – path to source data file(s). can use glob pattern
- **dst\_path** (str) – local folder or cloud storage (e.g., OCI object storage) prefix to save converted text files
- **encoding** (str, optional) – encoding for files, by default utf-8
- **storage\_options** (Dict, optional) – storage options for cloud storage, by default None

**Return type**

None

**engine**(*eng*: str) → None

Set engine for dataloader. Can be pandas or cudf.

**Parameters**

**eng** (str) – name of engine

**Return type**

None

**Raises**

**NotSupportedError** – raises error if engine passed in is not supported.

**metadata\_all**(*path*: str, *storage\_options*: Optional[Dict] = None, *encoding*: str = 'utf-8') → Generator[Dict[str, Any], None, None]

Get metadata of all files that matches the given path. Return a generator.

**Parameters**

- **path** (str) – path to data files. can use glob pattern.
- **storage\_options** (Dict, optional) – storage options for cloud storage, by default None
- **encoding** (str, optional) – encoding of files, by default 'utf-8'

**Returns**

generator of extracted metadata from files.

**Return type**

Generator

**metadata\_schema**(*path*: str, *n\_files*: int = 1, *storage\_options*: Optional[Dict] = None, *encoding*: str = 'utf-8') → List[str]

Get available fields in metadata by looking at the first *n\_files* that matches the given path.

#### Parameters

- **path** (str) – path to data files. can have glob pattern
- **n\_files** (int, optional) – number of files to look up, default to be 1
- **storage\_options** (dict, optional) – storage options for cloud storage, by default None
- **encoding** (str, optional) – encoding of files, by default utf-8

#### Returns

list of available fields in metadata

#### Return type

List[str]

**option**(*opt*: Options, *spec*: Optional[Any] = None) → None

Set extraction options.

#### Parameters

- **opt** (*ads.text\_dataset.options.Options*) – an option defined in *ads.text\_dataset.options.Options*
- **spec** (Any, optional) – specifications that will be passed to option handler, by default None

#### Return type

None

**read\_line**(*path*: str, *udf*: Union[str, Callable] = None, *n\_lines\_per\_file*: int = None, *total\_lines*: int = None, *df\_args*: Dict = None, *storage\_options*: Dict = None, *encoding*: str = 'utf-8') → Union[Generator[Union[str, List[str]], None, None], DataFrame]

Read each file into lines. If path matches multiple files, will combine lines from all files.

#### Parameters

- **path** (str) – path to data files. can have glob pattern.
- **udf** ((callable | str), optional) – user defined function for processing each line, can be a callable or regex, by default None
- **n\_lines\_per\_file** (int, optional) – max number of lines read from each file, by default None
- **total\_lines** (int, optional) – max number of lines read from all files, by default None
- **df\_args** (dict, optional) – arguments passed to dataframe engine (e.g. pandas), by default None
- **storage\_options** (dict, optional) – storage options for cloud storage, by default None
- **encoding** (str, optional) – encoding of files, by default 'utf-8'

#### Returns

returns either a data generator or a dataframe.

**Return type**

(Generator | DataFrame)

**read\_text**(*path*: str, *udf*: Union[str, Callable] = None, *total\_files*: int = None, *storage\_options*: Dict = None, *df\_args*: Dict = None, *encoding*: str = 'utf-8') → Union[Generator[Union[str, List[str]], None, None], DataFrame]

Read each file into a text string. If path matches multiple files, each file corresponds to one record.

**Parameters**

- **path** (*str*) – path to data files. can have glob pattern.
- **udf** ((*callable* | *str*), *optional*) – user defined function for processing each line, can be a callable or regex, by default None
- **total\_files** (*int*, *optional*) – max number of files to read, by default None
- **df\_args** (*dict*, *optional*) – arguments passed to dataframe engine (e.g. pandas), by default None
- **storage\_options** (*dict*, *optional*) – storage options for cloud storage, by default None
- **encoding** (*str*, *optional*) – encoding of files, by default 'utf-8'

**Returns**

returns either a data generator or a dataframe.

**Return type**

(Generator | DataFrame)

**with\_processor**(*processor\_type*: str) → None

Set file processor.

**Parameters**

**processor\_type** (*str*) – type of processor, which corresponds to format of the file.

**Return type**

None

**class** ads.text\_dataset.dataset.TextDatasetFactory

Bases: object

A class that generates a dataloader given a file format.

**static format**(*format\_name*: str) → [DataLoader](#)

Instantiates DataLoader class and seeds it with the right kind of FileProcessor. Eg. PDFProcessor for pdf. The FileProcessorFactory returns the processor based on the format Type.

**Parameters**

**format\_name** (*str*) – name of format

**Returns**

a [DataLoader](#) object.

**Return type**

*ads.text\_dataset.dataset.DataLoader*

#### 18.1.1.20.4 `ads.text_dataset.extractor` module

**class** `ads.text_dataset.extractor.FileProcessor`(*backend*: `Union[str, Base]` = 'default')

Bases: `object`

Base class for all the file processor. Files are opened using `fsspec` library. The default implementation in the base class assumes text files.

This class is expected to be used inside `ads.text_dataset.dataset.DataLoader`.

**backend**(*backend*: `Union[str, Base]`) → `None`

Set backend for file processor.

**Parameters**

**backend** (`ads.text_dataset.backends.Base`) – a backend for file processor

**Return type**

`None`

**Raises**

**NotSupportedError** – when specified backend is not supported.

**backend\_map** = {'default': <class 'ads.text\_dataset.backends.Base'>, 'tika': <class 'ads.text\_dataset.backends.Tika'>}

**convert\_to\_text**(*fhandler*: `OpenFile`, *dst\_path*: `str`, *fname*: `Optional[str]` = `None`, *storage\_options*: `Optional[Dict]` = `None`) → `str`

Convert input file to a text file.

**Parameters**

- **fhandler** (`fsspec.core.OpenFile`) – a file handler returned by `fsspec`
- **dst\_path** (`str`) – local folder or cloud storage (e.g. OCI object storage) prefix to save converted text files
- **fname** (`str`, *optional*) – filename for converted output, relative to dirname or prefix, by default `None`
- **storage\_options** (`dict`, *optional*) – storage options for cloud storage, by default `None`

**Returns**

path to saved output

**Return type**

`str`

**get\_metadata**(*fhandler*: `OpenFile`) → `Dict`

Get metadata of a file.

**Parameters**

**fhandler** (`fsspec.core.OpenFile`) – a file handler returned by `fsspec`

**Returns**

dictionary of metadata

**Return type**

`dict`

**read\_line**(*fhandler*: *OpenFile*, *\*\*format\_reader\_kwargs*: *Dict*) → Generator[Union[str, List[str]], None, None]

Yields lines from a file.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – file handler returned by *fsspec*

**Returns**

a generator that yields lines from a file

**Return type**

Generator

**read\_text**(*fhandler*: *OpenFile*, *\*\*format\_reader\_kwargs*: *Dict*) → Generator[Union[str, List[str]], None, None]

Yield contents from the entire file.

**Parameters**

**fhandler** (*fsspec.core.OpenFile*) – a file handler returned by *fsspec*

**Returns**

a generator that yield text from a file

**Return type**

Generator

**class** `ads.text_dataset.extractor.FileProcessorFactory`

Bases: `object`

Factory that manages all file processors. Provides functionality to get a processor corresponding to a given file type, or register custom processor for a specific file format.

## Examples

```
>>> from ads.text_dataset.extractor import FileProcessor, FileProcessorFactory
>>> FileProcessorFactory.get_processor('pdf')
>>> class CustomProcessor(FileProcessor):
...     # custom logic here
...     pass
>>> FileProcessorFactory.register('new_format', CustomProcessor)
```

**static** `get_processor(format)`

```
processor_map = {'doc': <class 'ads.text_dataset.extractor.WordProcessor'>, 'docx':
<class 'ads.text_dataset.extractor.WordProcessor'>, 'pdf': <class
'ads.text_dataset.extractor.PDFProcessor'>, 'txt': <class
'ads.text_dataset.extractor.FileProcessor'>}
```

**classmethod** `register(fmt: str, processor: FileProcessor) → None`

Register custom file processor for a file format.

**Parameters**

- **fmt** (*str*) – file format
- **processor** (*FileProcessor*) – custom processor

**Raises**

**TypeError** – raised when processor is not a subclass of *FileProcessor*.

```
class ads.text_dataset.extractor.PDFProcessor(backend: Union[str, Base] = 'default')
    Bases: FileProcessor
    Extracts text content from PDF

    backend_map = {'default': <class 'ads.text_dataset.backends.Tika'>, 'pdfplumber':
<class 'ads.text_dataset.backends.PDFPlumber'>, 'tika': <class
'ads.text_dataset.backends.Tika'>}
```

```
class ads.text_dataset.extractor.WordProcessor(backend: Union[str, Base] = 'default')
    Bases: FileProcessor
    Extracts text content from doc or docx format.

    backend_map = {'default': <class 'ads.text_dataset.backends.Tika'>, 'tika': <class
'ads.text_dataset.backends.Tika'>}
```

#### 18.1.1.20.5 ads.text\_dataset.options module

```
class ads.text_dataset.options.FileOption(dataloader: ads.text_dataset.dataset.DataLoader)
    Bases: OptionHandler
    handle(fhandler: OpenFile, spec: Any) → Any
```

```
class ads.text_dataset.options.MetadataOption(dataloader: ads.text_dataset.dataset.DataLoader)
    Bases: OptionHandler
    handle(fhandler: OpenFile, spec: Dict) → List
```

```
class ads.text_dataset.options.OptionFactory
    Bases: object
    static option_handler(option: Options) → OptionHandler

    option_handlers = {<Options.FILE_NAME: 1>: <class
'ads.text_dataset.options.FileOption'>, <Options.FILE_METADATA: 2>: <class
'ads.text_dataset.options.MetadataOption'>}
```

```
    classmethod register_option(option: Options, handler) → None
```

```
class ads.text_dataset.options.OptionHandler(dataloader: ads.text_dataset.dataset.DataLoader)
    Bases: object
    handle(fhandler: OpenFile, spec: Any) → Any
```

```
class ads.text_dataset.options.Options(value)
    Bases: Enum
    An enumeration.

    FILE_METADATA = 2

    FILE_NAME = 1
```

### 18.1.1.20.6 Module contents

### 18.1.1.21 ads.vault package

#### 18.1.1.21.1 Submodules

#### 18.1.1.21.2 ads.vault module

```
class ads.vault.vault.Vault(vault_id: Optional[str] = None, key_id: Optional[str] = None,
                             compartment_id=None, secret_client_auth=None, vault_client_auth=None,
                             auth=None)
```

Bases: object

##### Parameters

- **vault\_id** ((str, optional). Default None) – ocid of the vault
- **key\_id** ((str, optional). Default None) – ocid of the key that is used for encrypting the content
- **compartment\_id** ((str, optional). Default None) – ocid of the compartment\_id where the vault resides. When available in environment variable - `NB_SESSION_COMPARTMENT_OCID`, will default to that.
- **secret\_client\_auth** ((dict, optional, deprecated since 2.5.1). Default None.) – deprecated since 2.5.1. Use `auth` instead
- **vault\_client\_auth** ((dict, optional, deprecated since 2.5.1). Default None.) – deprecated since 2.5.1. Use `auth` instead
- **auth** ((dict, optional)) – Dictionary returned from `ads.common.auth.api_keys()` or `ads.common.auth.resource_principal()`. By default, will follow what is set in `ads.set_auth`. Use this attribute to override the default.

```
create_secret(value: dict, secret_name: Optional[str] = None, description: Optional[str] = None,
               encode=True, freeform_tags: Optional[dict] = None, defined_tags: Optional[dict] = None)
               → str
```

Saves value into vault as a secret.

##### Parameters

- **value** (dict) – The value to store as a secret.
- **secret\_name** (str, optional) – The name of the secret.
- **description** (str, optional) – The description of the secret.
- **encode** ((bool, optional). Default True) – Whether to encode using the default encoding.
- **freeform\_tags** ((dict, optional). Default None) – freeform\_tags as defined by the oci sdk
- **defined\_tags** ((dict, optional). Default None) – defined\_tags as defined by the oci sdk

##### Return type

The secret ocid that correspond to the value saved as a secret into vault.

**get\_secret**(*secret\_id*: str, *decoded*=True) → dict

Retrieve secret content based on the secret ocid provided

**Parameters**

- **secret\_id** (str) – The secret ocid.
- **decoded** ((bool, optional). Default True) – Whether to decode the content that is retrieved from vault service using the default decoder.

**Return type**

The secret content as a dictionary.

**update\_secret**(*secret\_id*: str, *secret\_content*: dict, *encode*: bool = True) → str

Updates content of a secret.

**Parameters**

- **secret\_id** (str) – The secret id where the stored secret will be updated.
- **secret\_content** (dict,) – The updated content.
- **encode** ((bool, optional). Default True) – Whether to encode the secret\_content using default encoding

**Return type**

The secret ocid with updated content.

#### 18.1.1.21.3 Module contents

### 18.1.2 Submodules

#### 18.1.3 ads.config module

**ads.config.open**(*uri*: Optional[str] = '~/.ads/config', *profile*: Optional[str] = 'DEFAULT', *mode*: Optional[str] = 'r', *auth*: Dict = None)

Context manager helping to read and write config files.

**Parameters**

- **uri** ((str, optional). Defaults to ~/.ads/config.) – The path to the config file. Can be local or Object Storage file.
- **profile** ((str, optional). Defaults to DEFAULT) – The name of the profile to be loaded.
- **mode** ((str, optional). Defaults to r.) – The config mode. Supported values: ['r', 'w']
- **auth** ((Dict, optional). Defaults to None.) – The default authentication is set using *ads.set\_auth* API. If you need to override the default, use the *ads.common.auth.api\_keys* or *ads.common.auth.resource\_principal* to create appropriate authentication signer and kwargs required to instantiate IdentityClient object.

**Yields**

*ConfigSection* – The config section object.



## 18.1.4 Module contents

`ads.getLogger(name='ads')`

`ads.hello()`

Imports Pandas, sets the documentation mode, and prints a fancy “Hello”.

`ads.set_debug_mode(mode=True)`

Enable/disable printing stack traces on notebook.

**Parameters**

**mode** (*bool* (*default True*)) – Enable/disable print stack traces on notebook

`ads.set_documentation_mode(mode=False)`

This method is deprecated and will be removed in future releases. Enable/disable printing user tips on notebook.

**Parameters**

**mode** (*bool* (*default False*)) – Enable/disable print user tips on notebook

`ads.set_expert_mode()`

This method is deprecated and will be removed in future releases. Enables the debug and documentation mode for expert users all in one method.

---

## Oracle Accelerated Data Science (ADS) SDK

The Oracle Accelerated Data Science (ADS) SDK is maintained by the Oracle Cloud Infrastructure Data Science service team. It speeds up common data science activities by providing tools that automate and/or simplify common data science tasks, along with providing a data scientist friendly pythonic interface to Oracle Cloud Infrastructure (OCI) services, most notably OCI Data Science, Data Flow, Object Storage, and the Autonomous Database. ADS gives you an interface to manage the lifecycle of machine learning models, from data acquisition to model evaluation, interpretation, and model deployment.

With ADS you can:

- Read datasets from Oracle Object Storage, Oracle RDBMS (ATP/ADW/On-prem), AWS S3, and other sources into Pandas dataframes.
  - Easily compute summary statistics on your dataframes and perform data profiling.
  - Tune models using hyperparameter optimization with the ADSTuner tool.
  - Generate detailed evaluation reports of your model candidates with the ADSEvaluator module.
  - Save machine learning models to the OCI Data Science Models.
  - Deploy those models as HTTPS endpoints with Model Deployment.
  - Launch distributed ETL, data processing, and model training jobs in Spark with OCI Data Flow.
  - Train machine learning models in OCI Data Science Jobs.
  - Manage the lifecycle of conda environments through the ads conda command line interface (CLI).
  - Distributed Training with PyTorch, Horovod and Dask
- 

## Installation

```
python3 -m pip install oracle-ads
```

---

---

## Source Code

<https://github.com/oracle/accelerated-data-science>

---

```
>>> import ads
>>> ads.hello()
```

```
  0  o-o  o-o
 / \ | \ |
o---o|  0 o-o
|   || /   |
o   oo-o o--o
```

```
ADS SDK version: X.Y.Z
Pandas version: x.y.z
Debug mode: False
```

## ADDITIONAL DOCUMENTATION

- [OCI Data Science and AI services Examples](#)
- [Oracle AI & Data Science Blog](#)
- [OCI Documentation](#)



## EXAMPLES

### 20.1 Load data from Object Storage

```
import ads
import oci
import pandas as pd

ads.set_auth(
    auth="api_key", oci_config_location=oci.config.DEFAULT_LOCATION, profile="DEFAULT"
)
bucket_name = "<bucket_name>"
path = "<path>"
namespace = "<namespace>"
df = pd.read_csv(
    f"oci://{bucket_name}@{namespace}/{path}", storage_options=ads.auth.default_signer()
)
```

### 20.2 Load data from Autonomous DB

This example uses SQL injection safe binding variables.

```
import ads
import pandas as pd

connection_parameters = {
    "user_name": "<user_name>",
    "password": "<password>",
    "service_name": "<tns_name>",
    "wallet_location": "<file_path>",
}

df = pd.DataFrame.ads.read_sql(
    """
    SELECT *
    FROM SH.SALES
    WHERE ROWNUM <= :max_rows
    """,
    bind_variables={ "max_rows" : 100 },
)
```

(continues on next page)

(continued from previous page)

```
connection_parameters=connection_parameters,  
)
```

## 20.3 More Examples

See *quick start* guide for more examples

## CONTRIBUTING

This project welcomes contributions from the community. Before submitting a pull request, please review our contribution guide [CONTRIBUTING.md](#).

Find Getting Started instructions for developers in [README-development.md](#)





## **SECURITY**

Consult the security guide [SECURITY.md](#) for our responsible security vulnerability disclosure process.



---

CHAPTER  
**TWENTYTHREE**

---

**LICENSE**

Copyright (c) 2020, 2022 Oracle and/or its affiliates. Licensed under the [Universal Permissive License v1.0](#)



## PYTHON MODULE INDEX

### a

- ads, 877
- ads.automl, 494
- ads.automl.driver, 487
- ads.automl.provider, 488
- ads.bds, 541
- ads.bds.auth, 539
- ads.catalog, 507
- ads.catalog.model, 494
- ads.catalog.notebook, 501
- ads.catalog.project, 503
- ads.catalog.summary, 506
- ads.common, 538
- ads.common.auth, 507
- ads.common.card\_identifier, 507
- ads.common.data, 516
- ads.common.decorator.deprecate, 523
- ads.common.decorator.runtime\_dependency, 521
- ads.common.function.fn\_util, 529
- ads.common.model, 518
- ads.common.model\_export\_util, 526
- ads.common.model\_introspect, 524
- ads.common.model\_metadata, 521
- ads.common.model\_metadata\_mixin, 538
- ads.common.utils, 529
- ads.config, 876
- ads.data\_labeling, 575
- ads.data\_labeling.boundingbox, 542
- ads.data\_labeling.constants, 544
- ads.data\_labeling.data\_labeling\_service, 545
- ads.data\_labeling.interface.loader, 541
- ads.data\_labeling.interface.parser, 541
- ads.data\_labeling.interface.reader, 541
- ads.data\_labeling.metadata, 546
- ads.data\_labeling.mixin.data\_labeling, 550
- ads.data\_labeling.ner, 548
- ads.data\_labeling.parser.export\_metadata\_parser, 553
- ads.data\_labeling.parser.export\_record\_parser, 553
- ads.data\_labeling.reader.dataset\_reader, 557
- ads.data\_labeling.reader.jsonl\_reader, 563
- ads.data\_labeling.reader.metadata\_reader, 564
- ads.data\_labeling.reader.record\_reader, 567
- ads.data\_labeling.record, 549
- ads.data\_labeling.visualizer.image\_visualizer, 569
- ads.data\_labeling.visualizer.text\_visualizer, 572
- ads.database, 576
- ads.database.connection, 575
- ads.dataflow, 584
- ads.dataflow.dataflow, 576
- ads.dataflow.dataflowssummary, 584
- ads.dataset, 623
- ads.dataset.classification\_dataset, 584
- ads.dataset.correlation, 588
- ads.dataset.correlation\_plot, 588
- ads.dataset.dataframe\_transformer, 590
- ads.dataset.dataset, 591
- ads.dataset.dataset\_browser, 602
- ads.dataset.dataset\_with\_target, 605
- ads.dataset.exception, 610
- ads.dataset.factory, 610
- ads.dataset.feature\_engineering\_transformer, 615
- ads.dataset.feature\_selection, 615
- ads.dataset.forecasting\_dataset, 616
- ads.dataset.helper, 616
- ads.dataset.label\_encoder, 619
- ads.dataset.pipeline, 619
- ads.dataset.plot, 619
- ads.dataset.progress, 620
- ads.dataset.recommendation, 620
- ads.dataset.recommendation\_transformer, 620
- ads.dataset.regression\_dataset, 621
- ads.dataset.sampled\_dataset, 621
- ads.dataset.target, 622
- ads.dataset.timeseries, 623
- ads.evaluations, 633
- ads.evaluations.evaluation\_plot, 623
- ads.evaluations.evaluator, 625
- ads.evaluations.statistical\_metrics, 631
- ads.feature\_engineering, 714

[ads.feature\\_engineering.accessor.dataframe\\_accessor](#), 689  
[ads.feature\\_engineering.accessor.mixin.correlation](#), 692  
[ads.feature\\_engineering.accessor.mixin.eda\\_mixin](#), 693  
[ads.feature\\_engineering.accessor.mixin.eda\\_mixin\\_series](#), 695  
[ads.feature\\_engineering.accessor.mixin.feature\\_types\\_mixin](#), 650  
[ads.feature\\_engineering.accessor.series\\_accessor](#), 700  
[ads.feature\\_engineering.adsstring.common\\_regex\\_mixin](#), 702  
[ads.feature\\_engineering.adsstring.oci\\_language](#), 703  
[ads.feature\\_engineering.adsstring.string](#), 653  
[ads.feature\\_engineering.exceptions](#), 633  
[ads.feature\\_engineering.feature\\_type.address](#), 653  
[ads.feature\\_engineering.feature\\_type.base](#), 656  
[ads.feature\\_engineering.feature\\_type.boolean](#), 657  
[ads.feature\\_engineering.feature\\_type.category](#), 659  
[ads.feature\\_engineering.feature\\_type.constant](#), 662  
[ads.feature\\_engineering.feature\\_type.continuous](#), 664  
[ads.feature\\_engineering.feature\\_type.creditcard](#), 666  
[ads.feature\\_engineering.feature\\_type.datetime](#), 670  
[ads.feature\\_engineering.feature\\_type.discrete](#), 673  
[ads.feature\\_engineering.feature\\_type.document](#), 675  
[ads.feature\\_engineering.feature\\_type.gis](#), 676  
[ads.feature\\_engineering.feature\\_type.handler.feature\\_handler\\_factory](#), 705  
[ads.feature\\_engineering.feature\\_type.handler.feature\\_handler](#), 710  
[ads.feature\\_engineering.feature\\_type.handler.watson\\_model\\_extractor](#), 713  
[ads.feature\\_engineering.feature\\_type.integer](#), 680  
[ads.feature\\_engineering.feature\\_type.ip\\_address](#), 682  
[ads.feature\\_engineering.feature\\_type.ip\\_address\\_v4](#), 684  
[ads.feature\\_engineering.feature\\_type.ip\\_address\\_v6](#), 686  
[ads.feature\\_engineering.feature\\_type.lat\\_long](#),  
[ads.feature\\_engineering.feature\\_type.object](#),  
[ads.feature\\_engineering.feature\\_type.ordinal](#),  
[ads.feature\\_engineering.feature\\_type.phone\\_number](#),  
[ads.feature\\_engineering.feature\\_type.string](#),  
[ads.feature\\_engineering.feature\\_type.text](#),  
[ads.feature\\_engineering.feature\\_type.unknown](#),  
[ads.feature\\_engineering.feature\\_type.zip\\_code](#),  
[ads.feature\\_engineering.feature\\_type\\_manager](#),  
[ads.hpo](#), 729  
[ads.hpo.distributions](#), 714  
[ads.hpo.search\\_cv](#), 717  
[ads.hpo.stopping\\_criterion](#), 728  
[ads.jobs](#), 763  
[ads.jobs.ads\\_job](#), 729  
[ads.jobs.builders.infrastructure.dataflow](#),  
[ads.jobs.builders.infrastructure.dsc\\_job](#), 753  
[ads.jobs.builders.runtimes.python\\_runtime](#),  
[ads.model](#), 800  
[ads.model.artifact](#), 763  
[ads.model.deployment](#), 813  
[ads.model.deployment.model\\_deployer](#), 800  
[ads.model.deployment.model\\_deployment](#), 804  
[ads.model.deployment.model\\_deployment\\_properties](#),  
[ads.model.extractor.automl\\_extractor](#), 791  
[ads.model.extractor.keras\\_extractor](#), 797  
[ads.model.extractor.lightgbm\\_extractor](#), 793  
[ads.model.extractor.model\\_info\\_extractor](#), 794  
[ads.model.extractor.model\\_info\\_extractor\\_factory](#),  
[ads.model.extractor.pytorch\\_extractor](#), 799  
[ads.model.extractor.sklearn\\_extractor](#), 795  
[ads.model.extractor.tensorflow\\_extractor](#), 798  
[ads.model.extractor.xgboost\\_extractor](#), 792  
[ads.model.framework](#), 843  
[ads.model.framework.automl\\_model](#), 813  
[ads.model.framework.lightgbm\\_model](#), 818  
[ads.model.framework.pytorch\\_model](#), 823  
[ads.model.framework.sklearn\\_model](#), 828  
[ads.model.framework.tensorflow\\_model](#), 833  
[ads.model.framework.xgboost\\_model](#), 838  
[ads.model.generic\\_model](#), 765  
[ads.model.model\\_properties](#), 788

- ads.model.runtime, [847](#)
- ads.model.runtime.env\_info, [843](#)
- ads.model.runtime.model\_deployment\_details,  
    [845](#)
- ads.model.runtime.model\_provenance\_details,  
    [845](#)
- ads.model.runtime.runtime\_info, [790](#)
- ads.model.runtime.utils, [846](#)
- ads.secrets, [865](#)
- ads.secrets.adb, [851](#)
- ads.secrets.auth\_token, [863](#)
- ads.secrets.big\_data\_service, [859](#)
- ads.secrets.mysqlldb, [855](#)
- ads.secrets.oracledb, [857](#)
- ads.secrets.secrets, [847](#)
- ads.text\_dataset, [875](#)
- ads.text\_dataset.backends, [865](#)
- ads.text\_dataset.dataset, [867](#)
- ads.text\_dataset.extractor, [872](#)
- ads.text\_dataset.options, [874](#)
- ads.vault, [876](#)
- ads.vault.vault, [875](#)





## A

ACCESS (*ads.model.deployment.model\_deployment.ModelDeploymentLogType* attribute), 809  
 access\_log (*ads.model.deployment.model\_deployment.ModelDeployment* property), 806  
 activate() (*ads.catalog.model.Model* method), 494, 495  
 ADBSecret (class in *ads.secrets.adb*), 851  
 ADBSecretKeeper (class in *ads.secrets.adb*), 851  
 add() (*ads.dataset.pipeline.TransformerPipeline* method), 619  
 add\_metrics() (*ads.evaluations.evaluator.ADSEvaluator* method), 626, 627  
 add\_models() (*ads.evaluations.evaluator.ADSEvaluator* method), 626, 628  
 address (*ads.feature\_engineering.adsstring.common\_regex\_mixin.CommonRegexMixin* property), 652  
 Address (class in *ads.feature\_engineering.feature\_type.address*), 653  
 ads  
     module, 877  
 ads.automl  
     module, 494  
 ads.automl.driver  
     module, 487  
 ads.automl.provider  
     module, 488  
 ads.bds  
     module, 541  
 ads.bds.auth  
     module, 539  
 ads.catalog  
     module, 507  
 ads.catalog.model  
     module, 494  
 ads.catalog.notebook  
     module, 501  
 ads.catalog.project  
     module, 503  
 ads.catalog.summary  
     module, 506  
 ads.common  
     module, 538  
     ads.common.auth  
         module, 507  
     ads.common.card\_identifier  
         module, 507  
     ads.common.data  
         module, 516  
     ads.common.decorator.deprecate  
         module, 523  
     ads.common.decorator.runtime\_dependency  
         module, 521  
     ads.common.function.fn\_util  
         module, 529  
     ads.common.model  
         module, 518  
     ads.common.model\_export\_util  
         module, 526  
     ads.common.model\_introspect  
         module, 524  
     ads.common.model\_metadata  
         module, 521  
     ads.common.model\_metadata\_mixin  
         module, 538  
     ads.common.utils  
         module, 529  
 ads.config  
     module, 876  
 ads.data\_labeling  
     module, 575  
 ads.data\_labeling.boundingBox  
     module, 542  
 ads.data\_labeling.constants  
     module, 544  
 ads.data\_labeling.data\_labeling\_service  
     module, 545  
 ads.data\_labeling.interface.loader  
     module, 541  
 ads.data\_labeling.interface.parser  
     module, 541  
 ads.data\_labeling.interface.reader  
     module, 541  
 ads.data\_labeling.metadata

module, 546	module, 615
ads.data_labeling.mixin.data_labeling	ads.dataset.feature_selection
module, 550	module, 615
ads.data_labeling.ner	ads.dataset.forecasting_dataset
module, 548	module, 616
ads.data_labeling.parser.export_metadata_parser	ads.dataset.helper
module, 553	module, 616
ads.data_labeling.parser.export_record_parser	ads.dataset.label_encoder
module, 553	module, 619
ads.data_labeling.reader.dataset_reader	ads.dataset.pipeline
module, 557	module, 619
ads.data_labeling.reader.jsonl_reader	ads.dataset.plot
module, 563	module, 619
ads.data_labeling.reader.metadata_reader	ads.dataset.progress
module, 564	module, 620
ads.data_labeling.reader.record_reader	ads.dataset.recommendation
module, 567	module, 620
ads.data_labeling.record	ads.dataset.recommendation_transformer
module, 549	module, 620
ads.data_labeling.visualizer.image_visualizer	ads.dataset.regression_dataset
module, 569	module, 621
ads.data_labeling.visualizer.text_visualizer	ads.dataset.sampled_dataset
module, 572	module, 621
ads.database	ads.dataset.target
module, 576	module, 622
ads.database.connection	ads.dataset.timeseries
module, 575	module, 623
ads.dataflow	ads.evaluations
module, 584	module, 633
ads.dataflow.dataflow	ads.evaluations.evaluation_plot
module, 576	module, 623
ads.dataflow.dataflowssummary	ads.evaluations.evaluator
module, 584	module, 625
ads.dataset	ads.evaluations.statistical_metrics
module, 623	module, 631
ads.dataset.classification_dataset	ads.feature_engineering
module, 584	module, 714
ads.dataset.correlation	ads.feature_engineering.accessor.dataframe_accessor
module, 588	module, 638
ads.dataset.correlation_plot	ads.feature_engineering.accessor.mixin.correlation
module, 588	module, 646
ads.dataset.dataframe_transformer	ads.feature_engineering.accessor.mixin.eda_mixin
module, 590	module, 646
ads.dataset.dataset	ads.feature_engineering.accessor.mixin.eda_mixin_series
module, 591	module, 649
ads.dataset.dataset_browser	ads.feature_engineering.accessor.mixin.feature_types_mixin
module, 602	module, 650
ads.dataset.dataset_with_target	ads.feature_engineering.accessor.series_accessor
module, 605	module, 643
ads.dataset.exception	ads.feature_engineering.adsstring.common_regex_mixin
module, 610	module, 652
ads.dataset.factory	ads.feature_engineering.adsstring.oci_language
module, 610	module, 653
ads.dataset.feature_engineering_transformer	ads.feature_engineering.adsstring.string

module, 653	module, 703
ads.feature_engineering.exceptions	ads.feature_engineering.feature_type_manager
module, 633	module, 634
ads.feature_engineering.feature_type.address	ads.hpo
module, 653	module, 729
ads.feature_engineering.feature_type.base	ads.hpo.distributions
module, 656	module, 714
ads.feature_engineering.feature_type.boolean	ads.hpo.search_cv
module, 657	module, 717
ads.feature_engineering.feature_type.category	ads.hpo.stopping_criterion
module, 659	module, 728
ads.feature_engineering.feature_type.constant	ads.jobs
module, 662	module, 763
ads.feature_engineering.feature_type.continuous	ads.jobs.ads_job
module, 664	module, 729
ads.feature_engineering.feature_type.creditcards	ads.jobs.builders.infrastructure.dataflow
module, 666	module, 744
ads.feature_engineering.feature_type.datetime	ads.jobs.builders.infrastructure.dsc_job
module, 670	module, 753
ads.feature_engineering.feature_type.discrete	ads.jobs.builders.runtimes.python_runtime
module, 673	module, 735
ads.feature_engineering.feature_type.document	ads.model
module, 675	module, 800
ads.feature_engineering.feature_type.gis	ads.model.artifact
module, 676	module, 763
ads.feature_engineering.feature_type.handler.feature_model_deployment	ads.model.deployment
module, 705	module, 813
ads.feature_engineering.feature_type.handler.feature_model_deployment.model_deployer	ads.model.deployment.model_deployer
module, 710	module, 800
ads.feature_engineering.feature_type.handler.watson_model_deployment	ads.model.deployment.model_deployment
module, 713	module, 804
ads.feature_engineering.feature_type.integer	ads.model.deployment.model_deployment_properties
module, 680	module, 809
ads.feature_engineering.feature_type.ip_address	ads.model.extractor.automl_extractor
module, 682	module, 791
ads.feature_engineering.feature_type.ip_address.pytorch	ads.model.extractor.keras_extractor
module, 684	module, 797
ads.feature_engineering.feature_type.ip_address.pytorch	ads.model.extractor.lightgbm_extractor
module, 686	module, 793
ads.feature_engineering.feature_type.lat_long	ads.model.extractor.model_info_extractor
module, 689	module, 794
ads.feature_engineering.feature_type.object	ads.model.extractor.model_info_extractor_factory
module, 692	module, 790
ads.feature_engineering.feature_type.ordinal	ads.model.extractor.pytorch_extractor
module, 693	module, 799
ads.feature_engineering.feature_type.phone_numbers	ads.model.extractor.sklearn_extractor
module, 695	module, 795
ads.feature_engineering.feature_type.string	ads.model.extractor.tensorflow_extractor
module, 698	module, 798
ads.feature_engineering.feature_type.text	ads.model.extractor.xgboost_extractor
module, 700	module, 792
ads.feature_engineering.feature_type.unknown	ads.model.framework
module, 702	module, 843
ads.feature_engineering.feature_type.zip_code	ads.model.framework.automl_model

module, 813  
ads.model.framework.lightgbm\_model  
  module, 818  
ads.model.framework.pytorch\_model  
  module, 823  
ads.model.framework.sklearn\_model  
  module, 828  
ads.model.framework.tensorflow\_model  
  module, 833  
ads.model.framework.xgboost\_model  
  module, 838  
ads.model.generic\_model  
  module, 765  
ads.model.model\_properties  
  module, 788  
ads.model.runtime  
  module, 847  
ads.model.runtime.env\_info  
  module, 843  
ads.model.runtime.model\_deployment\_details  
  module, 845  
ads.model.runtime.model\_provenance\_details  
  module, 845  
ads.model.runtime.runtime\_info  
  module, 790, 846  
ads.model.runtime.utils  
  module, 846  
ads.secrets  
  module, 865  
ads.secrets.adb  
  module, 851  
ads.secrets.auth\_token  
  module, 863  
ads.secrets.big\_data\_service  
  module, 859  
ads.secrets.mysqlldb  
  module, 855  
ads.secrets.oracledb  
  module, 857  
ads.secrets.secrets  
  module, 847  
ads.text\_dataset  
  module, 875  
ads.text\_dataset.backends  
  module, 865  
ads.text\_dataset.dataset  
  module, 867  
ads.text\_dataset.extractor  
  module, 872  
ads.text\_dataset.options  
  module, 874  
ads.vault  
  module, 876  
ads.vault.vault

  module, 875  
ADSDData (class in ads.common.data), 516  
ADSDDataFrameAccessor (class in  
  ads.feature\_engineering.accessor.dataframe\_accessor),  
  638  
ADSDataset (class in ads.dataset.dataset), 591  
ADSDatasetWithTarget (class in  
  ads.dataset.dataset\_with\_target), 605  
ADSEvaluator (class in ads.evaluations.evaluator), 625  
ADSEvaluator.EvaluationMetrics (class in  
  ads.evaluations.evaluator), 627  
ADSFeatureTypesMixin (class in  
  ads.feature\_engineering.accessor.mixin.feature\_types\_mixin),  
  651  
ADSModel (class in ads.common.model), 518  
ADSSeriesAccessor (class in  
  ads.feature\_engineering.accessor.series\_accessor),  
  643  
ADSSeriesValidator (class in  
  ads.feature\_engineering.accessor.series\_accessor),  
  646  
ADSTuner (class in ads.hpo.search\_cv), 717  
algorithm (ads.model.extractor.automl\_extractor.AutoMLExtractor  
  property), 791  
algorithm (ads.model.extractor.keras\_extractor.KerasExtractor  
  property), 797  
algorithm (ads.model.extractor.lightgbm\_extractor.LightgbmExtractor  
  property), 793  
algorithm (ads.model.extractor.pytorch\_extractor.PyTorchExtractor  
  property), 799  
algorithm (ads.model.extractor.sklearn\_extractor.SklearnExtractor  
  property), 796  
algorithm (ads.model.extractor.tensorflow\_extractor.TensorflowExtractor  
  property), 798  
algorithm (ads.model.extractor.xgboost\_extractor.XgboostExtractor  
  property), 792  
algorithm (ads.model.framework.automl\_model.AutoMLModel  
  attribute), 813  
algorithm (ads.model.framework.lightgbm\_model.LightGBMModel  
  attribute), 818  
algorithm (ads.model.framework.pytorch\_model.PyTorchModel  
  attribute), 823  
algorithm (ads.model.framework.sklearn\_model.SklearnModel  
  attribute), 828  
algorithm (ads.model.framework.tensorflow\_model.TensorFlowModel  
  attribute), 833  
algorithm (ads.model.framework.xgboost\_model.XGBoostModel  
  attribute), 838  
algorithm (ads.model.generic\_model.GenericModel at-  
  tribute), 768  
algorithm() (ads.model.extractor.lightgbm\_extractor.LightgbmExtractor  
  method), 793  
algorithm() (ads.model.extractor.model\_info\_extractor.ModelInfoExtract  
  method), 794

[algorithm\(\) \(ads.model.extractor.pytorch\\_extractor.PyTorchExtractor attribute\), 735](#)  
[method\), 799](#)  
[algorithm\(\) \(ads.model.extractor.sklearn\\_extractor.SklearnExtractor attribute\), 737](#)  
[method\), 796](#)  
[algorithm\(\) \(ads.model.extractor.tensorflow\\_extractor.TensorflowExtractor attribute\), 739](#)  
[method\), 798](#)  
[algorithm\(\) \(ads.model.extractor.xgboost\\_extractor.XgboostExtractor attribute\), 741](#)  
[method\), 792](#)  
[annotation \(ads.data\\_labeling.record.Record attribute\), 550](#)  
[annotation\\_type \(ads.data\\_labeling.metadata.Metadata attribute\), 547](#)  
[AnnotationType \(class in ads.data\\_labeling.constants\), 544](#)  
[API\\_KEY \(ads.common.auth.AuthType attribute\), 509](#)  
[api\\_keys\(\) \(in module ads.common.auth\), 511](#)  
[APIKey \(class in ads.common.auth\), 507](#)  
[application \(ads.jobs.builders.infrastructure.dataflow.DataFlow attribute\), 750](#)  
[archive\\_bucket \(ads.jobs.builders.runtimes.python\\_runtime.DataFlow attribute\), 737](#)  
[archive\\_uri \(ads.jobs.builders.runtimes.python\\_runtime.DataFlow attribute\), 737](#)  
[ArtifactFolderStructureError, 763](#)  
[artifact \(ads.jobs.builders.infrastructure.dsc\\_job.DSCJob attribute\), 754](#)  
[artifact\\_dir \(ads.model.framework.automl\\_model.AutoMLModel attribute\), 814](#)  
[artifact\\_dir \(ads.model.framework.lightgbm\\_model.LightGBMModel attribute\), 818](#)  
[artifact\\_dir \(ads.model.framework.pytorch\\_model.PyTorchModel attribute\), 823](#)  
[artifact\\_dir \(ads.model.framework.sklearn\\_model.SklearnModel attribute\), 828](#)  
[artifact\\_dir \(ads.model.framework.tensorflow\\_model.TensorFlowModel attribute\), 833](#)  
[artifact\\_dir \(ads.model.framework.xgboost\\_model.XGBoostModel attribute\), 838](#)  
[auth \(ads.model.framework.automl\\_model.AutoMLModel attribute\), 814](#)  
[auth \(ads.model.framework.lightgbm\\_model.LightGBMModel attribute\), 818](#)  
[auth \(ads.model.framework.pytorch\\_model.PyTorchModel attribute\), 823](#)  
[auth \(ads.model.framework.sklearn\\_model.SklearnModel attribute\), 828](#)  
[auth \(ads.model.framework.tensorflow\\_model.TensorFlowModel attribute\), 833](#)  
[auth \(ads.model.framework.xgboost\\_model.XGBoostModel attribute\), 838](#)  
[auth \(ads.model.generic\\_model.GenericModel attribute\), 768](#)  
[auth \(ads.secrets.big\\_data\\_service.BDSSecretKeeper attribute\), 862](#)  
[auth\\_token \(ads.secrets.auth\\_token.AuthToken attribute\), 863](#)  
[AuthFactory \(class in ads.common.auth\), 508](#)  
[AuthSignatureGenerator \(class in ads.common.auth\), 509](#)  
[AuthState \(class in ads.common.auth\), 509](#)  
[AuthToken \(class in ads.secrets.auth\\_token\), 863](#)  
[AuthTokenSecretKeeper \(class in ads.secrets.auth\\_token\), 863](#)  
[AuthType \(class in ads.common.auth\), 509](#)  
[auto\\_transform\(\) \(ads.dataset.classification\\_dataset.BinaryTextClassificationDataset method\), 585](#)  
[auto\\_transform\(\) \(ads.dataset.classification\\_dataset.ClassificationDataset method\), 585](#)  
[auto\\_transform\(\) \(ads.dataset.classification\\_dataset.MultiClassTextClassificationDataset method\), 588](#)  
[auto\\_transform\(\) \(ads.dataset.dataset\\_with\\_target.ADSDatasetWithTarget method\), 605](#)  
[AutoML \(class in ads.automl.driver\), 487](#)  
[AutoMLExtractor \(class in ads.model.extractor.automl\\_extractor\), 791](#)  
[AutoMLFeatureSelection \(class in ads.automl.provider\), 488](#)  
[AutoMLModel \(class in ads.model.framework.automl\\_model\), 813](#)  
[AutoMLPreprocessingTransformer \(class in ads.automl.provider\), 489](#)  
[AutoMLProvider \(class in ads.automl.provider\), 489](#)

AVAILABLE (*ads.model.generic\_model.ModelState* attribute), 787

## B

backend() (*ads.text\_dataset.dataset.DataLoader* method), 869

backend() (*ads.text\_dataset.extractor.FileProcessor* method), 872

backend\_map (*ads.text\_dataset.extractor.FileProcessor* attribute), 872

backend\_map (*ads.text\_dataset.extractor.PDFProcessor* attribute), 874

backend\_map (*ads.text\_dataset.extractor.WordProcessor* attribute), 874

Base (class in *ads.text\_dataset.backends*), 865

baseline (*ads.evaluations.evaluation\_plot.EvaluationPlot* attribute), 623

baseline\_kwargs (*ads.evaluations.evaluation\_plot.EvaluationPlot* attribute), 623, 624

BaselineAutoMLProvider (class in *ads.automl.provider*), 490

BaselineModel (class in *ads.automl.provider*), 491

batch\_convert\_case() (in module *ads.common.utils*), 530

BDS (*ads.common.decorator.runtime\_dependency.OptionalDependency* attribute), 522

BDSecret (class in *ads.secrets.big\_data\_service*), 859

BDSecretKeeper (class in *ads.secrets.big\_data\_service*), 860

best\_index (*ads.hpo.search\_cv.ADSTuner* property), 718

best\_params (*ads.hpo.search\_cv.ADSTuner* property), 718

best\_score (*ads.hpo.search\_cv.ADSTuner* property), 718

best\_scores() (*ads.hpo.search\_cv.ADSTuner* method), 718

BINARY\_CLASSIFICATION (*ads.common.utils.ml\_task\_types* attribute), 536

BINARY\_TEXT\_CLASSIFICATION (*ads.common.utils.ml\_task\_types* attribute), 536

BinaryClassificationDataset (class in *ads.dataset.classification\_dataset*), 584

BinaryTextClassificationDataset (class in *ads.dataset.classification\_dataset*), 585

block\_storage\_size (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun* property), 756

BokehHeatMap (class in *ads.dataset.correlation\_plot*), 588

Boolean (class in *ads.feature\_engineering.feature\_type.booleans*), 657

BOOSTED (*ads.common.decorator.runtime\_dependency.OptionalDependency* attribute), 522

bottom\_left (*ads.data\_labeling.boundingbox.BoundingBoxItem* attribute), 542

bottom\_right (*ads.data\_labeling.boundingbox.BoundingBoxItem* attribute), 542

BOUNDING\_BOX (*ads.data\_labeling.constants.AnnotationType* attribute), 544

BoundingBoxItem (class in *ads.data\_labeling.boundingbox*), 542

BoundingBoxItems (class in *ads.data\_labeling.boundingbox*), 543

BoundingBoxRecordParser (class in *ads.data\_labeling.parser.export\_record\_parser*), 553

boxes (*ads.data\_labeling.visualizer.image\_visualizer.LabeledImageItem* attribute), 571

branch (*ads.jobs.builders.runtimes.python\_runtime.GitPythonRuntime* property), 739

bucket\_uri (*ads.model.model\_properties.ModelProperties* attribute), 789

build() (*ads.common.data.ADSDData* static method), 516

build() (*ads.model.deployment.model\_deployment\_properties.ModelDeploymentProperties* method), 810, 811

## C

calculate\_cost() (*ads.evaluations.evaluator.ADSEvaluator* method), 626, 628

calculate\_sample\_size() (in module *ads.dataset.helper*), 616

call() (*ads.dataset.dataset.ADSDataset* method), 592

camel\_to\_snake() (in module *ads.common.utils*), 530

cancel() (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun* method), 762

card\_identify (class in *ads.common.card\_identifier*), 507

case (*ads.common.model\_introspect.PrintItem* attribute), 525

cat\_vs\_cat() (in module *ads.feature\_engineering.accessor.mixin.correlation*), 646

cat\_vs\_cont() (in module *ads.feature\_engineering.accessor.mixin.correlation*), 646

CategoricalDistribution (class in *ads.hpo.distributions*), 714

CategoricalLabel (*ads.feature\_engineering.feature\_type.category*), 659

CLASS (*ads.common.decorator.deprecate.TARGET\_TYPE* attribute), 523

classes (*ads.common.auth.AuthFactory* attribute), 508

classes (*ads.evaluations.statistical\_metrics.ModelEvaluator* attribute), 631



**ClassificationDataset** (class in `ads.dataset.classification_dataset`), 585  
**client** (`ads.jobs.builders.infrastructure.dataflow.DataFlowApp` attribute), 737  
   (`property`), 750  
**client** (`ads.jobs.builders.infrastructure.dataflow.DataFlowRun` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` attribute), 756  
   (`property`), 751  
**color\_wheel** (`ads.evaluations.evaluation_plot.EvaluationPlot` attribute), 623, 624  
**colors** (`ads.data_labeling.visualizer.image_visualizer.Renderer` attribute), 571  
**colors** (`ads.data_labeling.visualizer.text_visualizer.Renderer` attribute), 573  
**columns** (`ads.feature_engineering.accessor.dataframe_accessor` attribute), 638  
**commit** (`ads.jobs.builders.runtimes.python_runtime.GitPythonRuntime` attribute), 739  
   (`property`), 739  
**commit()** (`ads.catalog.model.Model` method), 494, 495  
**CommonRegexMixin** (class in `ads.feature_engineering.adsstring.common_regex`), 652  
**compartment\_id** (`ads.data_labeling.metadata.Metadata` attribute), 547  
**compartment\_id** (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` attribute), 737  
   (`property`), 756  
**compartment\_id** (`ads.model.model_properties.ModelProperties` attribute), 789  
**compartment\_id** (`ads.secrets.big_data_service.BDSecretKeeper` attribute), 862  
**COMPLETED** (`ads.hpo.search_cv.State` attribute), 728  
**compute()** (`ads.dataset.dataset.ADSDataset` method), 592  
**concatenate()** (in module `ads.dataset.helper`), 617  
**conda** (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` attribute), 735  
   (`property`), 735  
**CondaRuntime** (class in `ads.jobs.builders.runtimes.python_runtime`), 735  
**config** (`ads.dataflow.dataflow.DataFlowApp` `property`), 579  
**config** (`ads.dataflow.dataflow.DataFlowRun` `property`), 582  
**config** (`ads.dataflow.dataflow.RunObserver` `property`), 583  
**config** (`ads.model.deployment.model_deployer.ModelDeployer` attribute), 801  
**config** (`ads.model.deployment.model_deployment.ModelDeployment` attribute), 804  
**configuration** (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` attribute), 737  
   (`property`), 737  
**connect()** (`ads.database.connection.Connector` method), 575  
**Connector** (class in `ads.database.connection`), 575  
**CONST\_ARCHIVE\_BUCKET** (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` attribute), 736  
**CONST\_ARCHIVE\_URI** (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` attribute), 739  
**CONST\_BLOCK\_STORAGE** (`ads.jobs.builders.infrastructure.dataflow.DataFlowRuntime` attribute), 744  
**CONST\_BRANCH** (`ads.jobs.builders.runtimes.python_runtime.GitPythonRuntime` attribute), 739  
**CONST\_BUCKET\_URI** (`ads.jobs.builders.infrastructure.dataflow.DataFlowRuntime` attribute), 744  
**CONST\_COMMIT** (`ads.jobs.builders.runtimes.python_runtime.GitPythonRuntime` attribute), 739  
**CONST\_COMPARTMENT\_ID** (`ads.jobs.builders.infrastructure.dataflow.DataFlowRuntime` attribute), 744  
**CONST\_COMPARTMENT\_ID** (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` attribute), 756  
**CONST\_CONDA** (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` attribute), 735  
**CONST\_CONDA\_AUTH\_TYPE** (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` attribute), 737  
**CONST\_CONDA\_REGION** (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` attribute), 735  
**CONST\_CONDA\_SLUG** (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` attribute), 735  
**CONST\_CONDA\_TYPE** (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` attribute), 735  
**CONST\_CONDA\_TYPE\_CUSTOM** (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` attribute), 735  
**CONST\_CONDA\_TYPE\_SERVICE** (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` attribute), 735  
**CONST\_CONDA\_URI** (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` attribute), 735  
**CONST\_CONFIG** (`ads.jobs.builders.infrastructure.dataflow.DataFlowRuntime` attribute), 744  
**CONST\_CONFIGURATION** (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` attribute), 737  
**CONST\_DISPLAY\_NAME** (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` attribute), 756  
**CONST\_DRIVER\_SHAPE** (`ads.jobs.builders.infrastructure.dataflow.DataFlowRuntime` attribute), 744  
**CONST\_DRIVER\_SHAPE\_CONFIG** (`ads.jobs.builders.infrastructure.dataflow.DataFlowRuntime` attribute), 744  
**CONST\_ENTRYPOINT** (`ads.jobs.builders.runtimes.python_runtime.ScriptRuntime` attribute), 743  
**CONST\_EXCLUDE\_TAG** (`ads.jobs.builders.runtimes.python_runtime.NotebookRuntime` attribute), 741  
**CONST\_EXECUTE** (`ads.jobs.builders.infrastructure.dataflow.DataFlowRuntime` attribute), 744

attribute), 744  
 CONST\_EXECUTOR\_SHAPE (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_EXECUTOR\_SHAPE\_CONFIG (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_GIT\_SSH\_SECRET\_ID (ads.jobs.builders.runtimes.python\_runtime.GitPythonRuntime attribute), 739  
 CONST\_GIT\_URL (ads.jobs.builders.runtimes.python\_runtime attribute), 739  
 CONST\_ID (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_JOB\_INFRA (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_JOB\_TYPE (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_LANGUAGE (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_LOG\_GROUP\_ID (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_LOG\_ID (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_MEMORY\_IN\_GBS (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_MEMORY\_IN\_GBS (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_METASTORE\_ID (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_NOTEBOOK\_ENCODING (ads.jobs.builders.runtimes.python\_runtime.NotebookRuntime attribute), 741  
 CONST\_NOTEBOOK\_PATH (ads.jobs.builders.runtimes.python\_runtime.NotebookRuntime attribute), 741  
 CONST\_NUM\_EXECUTORS (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_OCPUS (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_OCPUS (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_OUTPUT\_URI (ads.jobs.builders.runtimes.python\_runtime.NotebookRuntime attribute), 741  
 CONST\_PROJECT\_ID (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_SCRIPT\_BUCKET (ads.jobs.builders.runtimes.python\_runtime.DataFlow attribute), 737  
 CONST\_SCRIPT\_PATH (ads.jobs.builders.runtimes.python\_runtime.ScriptRuntime attribute), 737  
 CONST\_SCRIPT\_PATH (ads.jobs.builders.runtimes.python\_runtime.ScriptRuntime attribute), 743  
 CONST\_SHAPE\_CONFIG\_DETAILS (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_SHAPE\_NAME (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_SKIP\_METADATA (ads.jobs.builders.runtimes.python\_runtime.GitPythonRuntime attribute), 739  
 CONST\_SPARK\_VERSION (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_SUBNET\_ID (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 756  
 CONST\_WAREHOUSE\_BUCKET\_URI (ads.jobs.builders.infrastructure.dataflow.DataFlow attribute), 744  
 CONST\_WORKING\_DIR (ads.jobs.builders.runtimes.python\_runtime.PythonRuntime attribute), 742  
 Correlation (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 662  
 cosine\_similarity() (in module ads.feature\_engineering.accessor.mixin.correlation), 646  
 content (ads.data\_labeling.record.Record attribute), 549, 550  
 Continuous (class in ads.feature\_engineering.feature\_type.continuous), 664  
 convert() (ads.jobs.builders.runtimes.python\_runtime.DataFlowNotebookRuntime attribute), 736  
 convert() (ads.jobs.builders.runtimes.python\_runtime.DataFlowRuntime attribute), 737  
 convert\_columns() (in module ads.dataset.helper), 617  
 convert\_dataframe\_schema() (ads.common.model.ADSModel static method), 518  
 convert\_to\_html() (in module ads.dataset.helper), 617  
 convert\_to\_text() (ads.text\_dataset.backends.Base method), 865  
 convert\_to\_text() (ads.text\_dataset.backends.PDFPlumber method), 866  
 convert\_to\_text() (ads.text\_dataset.backends.Tika method), 866  
 convert\_to\_text() (ads.text\_dataset.dataset.DataLoader attribute), 869  
 convert\_to\_text() (ads.text\_dataset.extractor.FileProcessor method), 872  
 convert\_to\_text\_classification() (ads.dataset.classification\_dataset.ClassificationDataset attribute), 531  
 copy\_file() (in module ads.common.utils), 531



- `copy_from_uri()` (in module `ads.common.utils`), 531  
`corr()` (`ads.dataset.dataset.ADSDataset` method), 592  
`correlation_ratio()` (`ads.feature_engineering.accessor.mixin.eda_mixin.EDADataAccessMixin` method), 646  
`correlation_ratio_plot()` (`ads.feature_engineering.accessor.mixin.eda_mixin.EDADataAccessMixin` method), 647  
`cramersv()` (`ads.feature_engineering.accessor.mixin.eda_mixin.EDADataAccessMixin` method), 647  
`cramersv_plot()` (`ads.feature_engineering.accessor.mixin.eda_mixin.EDADataAccessMixin` method), 647  
`create()` (`ads.jobs.ads_job.Job` method), 731  
`create()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 745  
`create()` (`ads.jobs.builders.infrastructure.dataflow.DataFlowApp` method), 750  
`create()` (`ads.jobs.builders.infrastructure.dataflow.DataFlowRun` method), 751  
`create()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJobRun` method), 756  
`create()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJobRun` method), 762  
`create()` (`ads.jobs.builders.infrastructure.dsc_job.DSCJobRun` method), 754  
`create_app()` (`ads.dataflow.dataflow.DataFlow` method), 576  
`create_notebook_session()` (`ads.catalog.notebook.NotebookCatalog` method), 501  
`create_project()` (`ads.catalog.project.ProjectCatalog` method), 503  
`create_secret()` (`ads.vault.vault.Vault` method), 875  
`create_signer()` (`ads.common.auth.APIKey` method), 508  
`create_signer()` (`ads.common.auth.AuthSignerGenerator` method), 509  
`create_signer()` (`ads.common.auth.InstancePrincipal` method), 510  
`create_signer()` (`ads.common.auth.ResourcePrincipal` method), 511  
`create_signer()` (in module `ads.common.auth`), 512  
`credit_card` (`ads.feature_engineering.adsstring.common_regex_mixin.CommonRegexMixin` property), 652  
`CreditCard` (class in `ads.feature_engineering.feature_type.creditcard`), 666  
`CustomFormatReaders` (class in `ads.dataset.factory`), 610
- D**  
`DATA` (`ads.common.decorator.runtime_dependency.OptionalDependency` attribute), 522  
`database` (`ads.secrets.mysqladb.MySQLDBSecret` attribute), 855  
`DataFlow` (class in `ads.dataflow.dataflow`), 576  
`DataFlow` (class in `ads.jobs.builders.infrastructure.dataflow`), 744  
`DataFlowMixin` (class in `ads.jobs.builders.infrastructure.dataflow`), 731  
`DataFlowApp` (class in `ads.dataflow.dataflow`), 578  
`DataFlowApp` (class in `ads.jobs.builders.infrastructure.dataflow`), 736  
`DataFlowLog` (class in `ads.dataflow.dataflow`), 580  
`DataFlowRun` (class in `ads.dataflow.dataflow`), 582  
`DataFlowRun` (class in `ads.jobs.builders.infrastructure.dataflow`), 750  
`DataFlowNotebookRuntime` (class in `ads.jobs.builders.runtimes.python_runtime`), 736  
`DataFrameLabelEncoder` (class in `ads.dataset.label_encoder`), 619  
`DataFrameTransformer` (class in `ads.dataset.dataframe_transformer`), 590  
`DataLabeling` (class in `ads.data_labeling.data_labeling_service`), 545  
`DataLabelingAccessMixin` (class in `ads.data_labeling.mixin.data_labeling`), 550  
`DataLoader` (class in `ads.text_dataset.dataset`), 867  
`datascience_job()` (`ads.jobs.ads_job.Job` static method), 731  
`DataScienceJob` (class in `ads.jobs.builders.infrastructure.dsc_job`), 755  
`DataScienceJobRun` (class in `ads.jobs.builders.infrastructure.dsc_job`), 761  
`DataScienceModelType` (class in `ads.common.debugging.debugging_mixin.CommonDebuggingMixin`), 765  
`dataset_id` (`ads.data_labeling.metadata.Metadata` attribute), 547  
`dataset_name` (`ads.data_labeling.metadata.Metadata` attribute), 547  
`dataset_type` (`ads.data_labeling.metadata.Metadata` attribute), 547  
`DatasetBrowser` (class in `ads.dataset.dataset_browser`), 602  
`DatasetDefaults` (class in `ads.dataset.helper`), 616  
`DatasetError`, 610  
`DatasetFactory` (class in `ads.dataset.factory`), 611

[DatasetLoadException](#), 616  
[DatasetNotFoundError](#), 565  
[DatasetType](#) (class in [ads.data\\_labeling.constants](#)), 544  
[date](#) ([ads.feature\\_engineering.adsstring.common\\_regex\\_mixin.CommonRegexMixin](#) property), 652  
[DateTime](#) (class in [ads.feature\\_engineering.feature\\_type.datetime](#)), 670  
[datetime\\_format](#) ([ads.catalog.project.ProjectSummaryList](#) attribute), 505  
[ddf](#) ([ads.dataset.dataset.ADSDataset](#) property), 593  
[deactivate\(\)](#) ([ads.catalog.model.Model](#) method), 494, 495  
[debug\(\)](#) ([ads.dataset.correlation\\_plot.BokehHeatMap](#) method), 588  
[decide\\_estimator\(\)](#) ([ads.automl.provider.BaselineAutoMLProvider](#) method), 490  
[decode\(\)](#) ([ads.secrets.adb.ADBSecretKeeper](#) method), 854  
[decode\(\)](#) ([ads.secrets.auth\\_token.AuthTokenSecretKeeper](#) method), 864  
[decode\(\)](#) ([ads.secrets.big\\_data\\_service.BDSSecretKeeper](#) method), 862  
[decode\(\)](#) ([ads.secrets.mysqladb.MySQLDBSecretKeeper](#) method), 856  
[decode\(\)](#) ([ads.secrets.oracledb.OracleDBSecretKeeper](#) method), 858  
[decode\(\)](#) ([ads.secrets.secrets.SecretKeeper](#) method), 849  
[decode\(\)](#) (in module [ads.hpo.distributions](#)), 716  
[default\(\)](#) ([ads.common.utils.JsonConverter](#) method), 530  
[default\(\)](#) ([ads.hpo.distributions.DistributionEncode](#) method), 715  
[default\\_color](#) ([ads.data\\_labeling.visualizer.image\\_visualizer.RenderOptions](#) attribute), 571  
[default\\_color](#) ([ads.data\\_labeling.visualizer.text\\_visualizer.RenderOptions](#) attribute), 573  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.address](#)), 656  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.boolean](#)), 659  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.creditcard](#)), 670  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.datetime](#)), 672  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.gis](#)), 679  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.ip\\_address](#)), 684  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.ip\\_address\\_v4](#)), 684  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.ip\\_address\\_v6](#)), 688  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.lat\\_long](#)), 692  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.phone\\_number](#)), 697  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.string](#)), 700  
[default\\_handler\(\)](#) (in module [ads.feature\\_engineering.feature\\_type.zip\\_code](#)), 705  
[DEFAULT\\_INFRA\\_TYPE](#) ([ads.jobs.builders.infrastructure.dsc\\_job.DSCJob](#) attribute), 754  
[DEFAULT\\_LABELS\\_MAP](#) ([ads.evaluations.evaluator.ADSEvaluator.EvaluationOptions](#) attribute), 627  
[default\\_signer\(\)](#) (in module [ads.common.auth](#)), 513  
[DEFAULT\\_SQL\\_ARRAYSIZE](#) ([ads.dataset.factory.CustomFormatReaders](#) attribute), 610  
[DEFAULT\\_SQL\\_CHUNKSIZE](#) ([ads.dataset.factory.CustomFormatReaders](#) attribute), 610  
[DEFAULT\\_SQL\\_CTU](#) ([ads.dataset.factory.CustomFormatReaders](#) attribute), 610  
[DEFAULT\\_SQL\\_MIL](#) ([ads.dataset.factory.CustomFormatReaders](#) attribute), 610  
[default\\_type](#) ([ads.feature\\_engineering.accessor.dataframe\\_accessor.ADSSeriesAccessor](#) property), 640  
[default\\_type](#) ([ads.feature\\_engineering.accessor.dataframe\\_accessor.ADSSeriesAccessor](#) property), 640  
[default\\_type](#) ([ads.feature\\_engineering.accessor.series\\_accessor.ADSSeriesAccessor](#) attribute), 644  
[default\\_type](#) ([ads.feature\\_engineering.accessor.series\\_accessor.ADSSeriesAccessor](#) property), 644  
[del\\_metrics\(\)](#) ([ads.evaluations.evaluator.ADSEvaluator](#) method), 626, 629  
[del\\_models\(\)](#) ([ads.evaluations.evaluator.ADSEvaluator](#) method), 626, 629  
[delete\(\)](#) ([ads.jobs.ads\\_job.Job](#) method), 731  
[delete\(\)](#) ([ads.jobs.builders.infrastructure.dataflow.DataFlow](#) method), 745  
[delete\(\)](#) ([ads.jobs.builders.infrastructure.dataflow.DataFlowApp](#) method), 750  
[delete\(\)](#) ([ads.jobs.builders.infrastructure.dataflow.DataFlowRun](#) method), 751  
[delete\(\)](#) ([ads.jobs.builders.infrastructure.dsc\\_job.DataScienceJob](#)

`method`), 756  
`delete()` (`ads.jobs.builders.infrastructure.dsc_job.DSCJob` `method`), 754  
`delete()` (`ads.model.deployment.model_deployer.ModelDeployer` `method`), 801, 802  
`delete()` (`ads.model.deployment.model_deployment.ModelDeployment` `method`), 806  
`delete()` (`ads.model.generic_model.GenericModel` `class method`), 771  
`delete_deployment()` (`ads.model.framework.automl_model.AutoMLModel` `method`), 815  
`delete_deployment()` (`ads.model.framework.lightgbm_model.LightGBMModel` `method`), 820  
`delete_deployment()` (`ads.model.framework.pytorch_model.PyTorchModel` `method`), 825  
`delete_deployment()` (`ads.model.framework.sklearn_model.SklearnModel` `method`), 830  
`delete_deployment()` (`ads.model.framework.tensorflow_model.TensorFlowModel` `method`), 835  
`delete_deployment()` (`ads.model.framework.xgboost_model.XGBoostModel` `method`), 840  
`delete_deployment()` (`ads.model.generic_model.GenericModel` `method`), 770, 772  
`delete_model()` (`ads.catalog.model.ModelCatalog` `method`), 496, 497  
`delete_notebook_session()` (`ads.catalog.notebook.NotebookCatalog` `method`), 502  
`delete_project()` (`ads.catalog.project.ProjectCatalog` `method`), 504  
`deploy()` (`ads.model.deployment.model_deployer.ModelDeployer` `method`), 801, 802  
`deploy()` (`ads.model.deployment.model_deployment.ModelDeployment` `method`), 805, 806  
`deploy()` (`ads.model.framework.automl_model.AutoMLModel` `method`), 816  
`deploy()` (`ads.model.framework.lightgbm_model.LightGBMModel` `method`), 820  
`deploy()` (`ads.model.framework.pytorch_model.PyTorchModel` `method`), 825  
`deploy()` (`ads.model.framework.sklearn_model.SklearnModel` `method`), 830  
`deploy()` (`ads.model.framework.tensorflow_model.TensorFlowModel` `method`), 835  
`deploy()` (`ads.model.framework.xgboost_model.XGBoostModel` `method`), 840  
`deploy()` (`ads.model.generic_model.GenericModel` `method`), 770, 772  
`deploy_from_model_uri()` (`ads.model.deployment.model_deployer.ModelDeployer` `method`), 802  
`deployment_access_log_id` (`ads.model.model_properties.ModelProperties` `attribute`), 789  
`deployment_bandwidth_mbps` (`ads.model.model_properties.ModelProperties` `attribute`), 789  
`deployment_instance_count` (`ads.model.model_properties.ModelProperties` `attribute`), 789  
`deployment_instance_shape` (`ads.model.model_properties.ModelProperties` `attribute`), 789  
`deployment_log_group_id` (`ads.model.model_properties.ModelProperties` `attribute`), 789  
`deployment_memory_in_gbs` (`ads.model.model_properties.ModelProperties` `attribute`), 789  
`deployment_ocpus` (`ads.model.model_properties.ModelProperties` `attribute`), 789  
`deployment_predict_log_id` (`ads.model.model_properties.ModelProperties` `attribute`), 789  
`deprecate_default_value()` (in `ads.dataset.helper` `module`), 617  
`deprecate_variable()` (in `ads.dataset.helper` `module`), 617  
`deprecated()` (in `ads.common.decorator.deprecate` `module`), 523  
`description` (`ads.feature_engineering.feature_type.address.Address` `attribute`), 653, 654  
`description` (`ads.feature_engineering.feature_type.base.FeatureType` `attribute`), 656  
`description` (`ads.feature_engineering.feature_type.boolean.Boolean` `attribute`), 657, 658  
`description` (`ads.feature_engineering.feature_type.category.Category` `attribute`), 659, 660  
`description` (`ads.feature_engineering.feature_type.constant.Constant` `attribute`), 662  
`description` (`ads.feature_engineering.feature_type.continuous.Continuous` `attribute`), 664  
`description` (`ads.feature_engineering.feature_type.creditcard.CreditCard` `attribute`), 666, 667  
`description` (`ads.feature_engineering.feature_type.datetime.DateTime` `attribute`), 670, 671  
`description` (`ads.feature_engineering.feature_type.discrete.Discrete` `attribute`), 673  
`description` (`ads.feature_engineering.feature_type.document.Document` `attribute`), 675  
`description` (`ads.feature_engineering.feature_type.gis.GIS` `attribute`), 675

attribute), 676, 677  
 description(ads.feature\_engineering.feature\_type.integer.Integer attribute), 680  
 description(ads.feature\_engineering.feature\_type.ip\_address.IpAddress attribute), 682, 683  
 description(ads.feature\_engineering.feature\_type.ip\_address.IpAddress attribute), 684, 685  
 description(ads.feature\_engineering.feature\_type.ip\_address.IpAddress attribute), 686, 687  
 description(ads.feature\_engineering.feature\_type.lat\_long.LatLong attribute), 689, 690  
 description(ads.feature\_engineering.feature\_type.object.Object attribute), 692, 693  
 description(ads.feature\_engineering.feature\_type.ordinal.Ordinal attribute), 693, 694  
 description(ads.feature\_engineering.feature\_type.phone\_number.PhoneNumber attribute), 696  
 description(ads.feature\_engineering.feature\_type.string.String attribute), 698  
 description(ads.feature\_engineering.feature\_type.text.Text attribute), 700, 701  
 description(ads.feature\_engineering.feature\_type.unknown.Unknown attribute), 702  
 description(ads.feature\_engineering.feature\_type.zip\_code.ZipCode attribute), 703  
 detect\_encoding() (ads.text\_dataset.backends.Tika method), 867  
 df (ads.catalog.project.ProjectSummaryList attribute), 505  
 df\_read\_functions (ads.dataset.dataset.ADSDataset attribute), 593  
 Discrete (class in ads.feature\_engineering.feature\_type.discrete), 673  
 DiscreteUniformDistribution (class in ads.hpo.distributions), 714  
 Distribution (class in ads.hpo.distributions), 714  
 DistributionEncode (class in ads.hpo.distributions), 714  
 DLSDatasetReader (class in ads.data\_labeling.reader.dataset\_reader), 558  
 DLSMetadataReader (class in ads.data\_labeling.reader.metadata\_reader), 564  
 DOCUMENT (ads.data\_labeling.constants.DatasetType attribute), 544  
 Document (class in ads.feature\_engineering.feature\_type.document.Document), 675  
 DONE (ads.model.generic\_model.ModelState attribute), 787  
 double\_overlay\_plots (ads.evaluations.evaluation\_plot.EvaluationPlot attribute), 624  
 down\_sample() (ads.dataset.classification\_dataset.ClassificationDataset attribute), 611  
 down\_sample() (in module ads.dataset.helper), 617  
 download() (ads.dataset.factory.DatasetFactory static method), 611  
 download() (ads.jobs.ads\_job.Job method), 731  
 download\_data() (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun method), 762  
 download\_data\_from\_web() (ads.jobs.builders.infrastructure.dsc\_job.DSCJob method), 754  
 download\_from\_web() (in module ads.common.utils), 532  
 download\_model() (ads.catalog.model.ModelCatalog attribute), 496, 497  
 driver (ads.jobs.builders.infrastructure.dataflow.DataFlowLogs attribute), 550  
 drop\_columns() (ads.dataset.dataset.ADSDataset method), 593  
 ds\_client (ads.model.deployment.model\_deployer.ModelDeployer attribute), 801  
 ds\_client (ads.model.deployment.model\_deployment.ModelDeployment attribute), 805  
 ds\_client (ads.model.framework.automl\_model.AutoMLModel attribute), 814  
 ds\_client (ads.model.framework.lightgbm\_model.LightGBMModel attribute), 818  
 ds\_client (ads.model.framework.pytorch\_model.PyTorchModel attribute), 823  
 ds\_client (ads.model.framework.sklearn\_model.SklearnModel attribute), 828  
 ds\_client (ads.model.framework.tensorflow\_model.TensorFlowModel attribute), 833  
 ds\_client (ads.model.framework.xgboost\_model.XGBoostModel attribute), 838  
 ds\_composite\_client (ads.model.deployment.model\_deployer.ModelDeployer attribute), 801  
 ds\_composite\_client (ads.model.deployment.model\_deployment.ModelDeployment attribute), 805  
 DSCJob (class in ads.jobs.builders.infrastructure.dsc\_job), 753  
 DSCJobRun (in module ads.jobs.builders.infrastructure.dsc\_job), 755  
 dsn (ads.secrets.oracledb.OracleDBSecret attribute), 857  
 DummyProgressBar (class in ads.dataset.progress), 620  
 DuplicatedStudyError, 727  
**E**  
 EDAMixin (class in ads.feature\_engineering.accessor.mixin.eda\_mixin), 646  
 EDAMixinSeries (class in ads.feature\_engineering.accessor.mixin.eda\_mixin\_series), 646



649

`ElaboratedPath` (class in `ads.dataset.helper`), 616

`ellipsis_strings()` (in module `ads.common.utils`), 532

`email` (`ads.feature_engineering.adsstring.common_regex_mixin.RegexMixin` (class in `ads.feature_engineering.adsstring.common_regex_mixin`), 652

`EmptyMetadata`, 565

`encode()` (`ads.secrets.adb.ADBSecretKeeper` method), 854

`encode()` (`ads.secrets.big_data_service.BDSecretKeeper` method), 863

`encode()` (`ads.secrets.secrets.SecretKeeper` method), 849

`encode()` (in module `ads.hpo.distributions`), 716

`engine()` (`ads.text_dataset.dataset.DataLoader` method), 869

`ENTITY_EXTRACTION` (`ads.data_labeling.constants.AnnotationType` attribute), 553

`EntityType` (class in `ads.data_labeling.parser.export_record_parser.ExportRecordParser`), 554

`entrypoint` (`ads.jobs.builders.runtimes.python_runtime.ScipyRuntime` (class in `ads.jobs.builders.runtimes.python_runtime`), 743

`ents` (`ads.data_labeling.visualizer.text_visualizer.LabeledTextVisualizer` (class in `ads.data_labeling.visualizer`), 573

`EnvInfo` (class in `ads.model.runtime.env_info`), 843

`est` (`ads.automl.provider.AutoMLProvider` property), 489

`estimator` (`ads.model.extractor.automl_extractor.AutoMLExtractor` (class in `ads.model.extractor`), 791

`estimator` (`ads.model.extractor.keras_extractor.KerasExtractor` (class in `ads.model.extractor`), 797

`estimator` (`ads.model.extractor.lightgbm_extractor.LightGBMExtractor` (class in `ads.model.extractor`), 793

`estimator` (`ads.model.extractor.pytorch_extractor.PyTorchExtractor` (class in `ads.model.extractor`), 799

`estimator` (`ads.model.extractor.sklearn_extractor.SklearnExtractor` (class in `ads.model.extractor`), 795

`estimator` (`ads.model.extractor.tensorflow_extractor.TensorFlowExtractor` (class in `ads.model.extractor`), 798

`estimator` (`ads.model.extractor.xgboost_extractor.XgboostExtractor` (class in `ads.model.extractor`), 792

`estimator` (`ads.model.framework.automl_model.AutoMLModel` (class in `ads.model.framework`), 814

`estimator` (`ads.model.framework.lightgbm_model.LightGBMModel` (class in `ads.model.framework`), 818

`estimator` (`ads.model.framework.pytorch_model.PyTorchModel` (class in `ads.model.framework`), 823

`estimator` (`ads.model.framework.sklearn_model.SklearnModel` (class in `ads.model.framework`), 828

`estimator` (`ads.model.framework.tensorflow_model.TensorFlowModel` (class in `ads.model.framework`), 833

`estimator` (`ads.model.framework.xgboost_model.XGBoostModel` (class in `ads.model.framework`), 839

`estimator` (`ads.model.generic_model.GenericModel` (class in `ads.model`), 768

`ev_test` (`ads.evaluations.evaluator.ADSEvaluator.EvaluationMetrics` (class in `ads.evaluations.evaluator`), 627

`ev_train` (`ads.evaluations.evaluator.ADSEvaluator.EvaluationMetrics` (class in `ads.evaluations.evaluator`), 627

`evaluation_plot` (`ads.evaluations.evaluator.ADSEvaluator.EvaluationMetrics` (class in `ads.evaluations.evaluator`), 623

`evaluations` (`ads.evaluations.evaluator.ADSEvaluator` (class in `ads.evaluations`), 625

`exclude_tag` (`ads.jobs.builders.runtimes.python_runtime.NotebookRuntime` (class in `ads.jobs.builders.runtimes.python_runtime`), 741

`executor` (`ads.jobs.builders.infrastructure.dataflow.DataFlowLogs` (class in `ads.jobs.builders.infrastructure.dataflow`), 750

`ExitCriterionError`, 727

`expand_lambda_function()` (in module `ads.dataset.dataframe_transformer`), 590

`EXPECTED_KEYS` (`ads.data_labeling.parser.export_metadata_parser.MetadataParser` (class in `ads.data_labeling.parser`), 545

`export()` (`ads.data_labeling.data_labeling_service.DataLabelingService` (class in `ads.data_labeling.data_labeling_service`), 545

`export_dict()` (`ads.secrets.secrets.Secret` method), 847

`export_options()` (`ads.secrets.secrets.Secret` method), 848

`export_vault_details()` (`ads.secrets.secrets.SecretKeeper` (class in `ads.secrets.secrets`), 849

`ExportMetadataReader` (class in `ads.data_labeling.reader.metadata_reader`), 565

`ExportReader` (class in `ads.data_labeling.reader.dataset_reader`), 569

`extract_info()` (`ads.model.extractor.model_info_extractor_factory.ModelInfoExtractorFactory` (class in `ads.model.extractor.model_info_extractor_factory`), 559

`extract_lib_dependencies_from_model()` (in module `ads.common.utils`), 532

**F**

`failures` (`ads.common.model_introspect.ModelIntrospect` (class in `ads.common.model_introspect`), 525

`feature_count()` (`ads.feature_engineering.accessor.mixin.eda_mixin.EDAAccessorMixin` (class in `ads.feature_engineering.accessor.mixin`), 647

`feature_domain()` (`ads.feature_engineering.feature_type.address.AddressFeatureType` (class in `ads.feature_engineering.feature_type.address`), 654

`feature_domain()` (`ads.feature_engineering.feature_type.boolean.BooleanFeatureType` (class in `ads.feature_engineering.feature_type.boolean`), 658

`feature_domain()` (`ads.feature_engineering.feature_type.category.CategoryFeatureType` (class in `ads.feature_engineering.feature_type.category`), 660

`feature_domain()` (`ads.feature_engineering.feature_type.constant.ConstantFeatureType` (class in `ads.feature_engineering.feature_type.constant`), 662

`feature_domain()` (`ads.feature_engineering.feature_type.continuous.ContinuousFeatureType` (class in `ads.feature_engineering.feature_type.continuous`), 664

`feature_domain()` (`ads.feature_engineering.feature_type.creditcard.CreditCardFeatureType` (class in `ads.feature_engineering.feature_type.creditcard`), 667

**feature\_domain()** (*ads.feature\_engineering.feature\_type.continuous.Continuous*  
class method), 671  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.continuous.Continuous*  
class method), 673  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.creditcard.CreditCard*  
class method), 675  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.creditcard.CreditCard*  
class method), 677  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.datetime.DateTime*  
class method), 680  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.datetime.DateTime*  
class method), 683  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.discrete.Discrete*  
class method), 685  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.discrete.Discrete*  
class method), 687  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.gis.GIS*  
class method), 690  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.gis.GIS*  
class method), 693  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.integer.Integer*  
class method), 694  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.integer.Integer*  
class method), 696  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.lat\_long.LatLong*  
class method), 698  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.lat\_long.LatLong*  
class method), 701  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.ordinal.Ordinal*  
class method), 702  
**feature\_domain()** (*ads.feature\_engineering.feature\_type.ordinal.Ordinal*  
class method), 704  
**feature\_names()** (*ads.common.model.ADSModel* class method), 518  
**feature\_plot()** (*ads.feature\_engineering.accessor.mixin.eda\_mixin.EDA\_Mixin*  
method), 648  
**feature\_plot()** (*ads.feature\_engineering.accessor.mixin.eda\_mixin.EDA\_Mixin*  
method), 649  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.address.Address*  
static method), 654  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.zip\_code.ZipCode*  
static method), 655  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.zip\_code.ZipCode*  
method), 657  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.zip\_code.ZipCode*  
static method), 658  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.category.Category*  
method), 660  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.category.Category*  
static method), 660  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.address.Address*  
method), 662  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.address.Address*  
static method), 663  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.continuous.Continuous*  
method), 664  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.continuous.Continuous*  
static method), 665  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.creditcard.CreditCard*  
method), 667  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.creditcard.CreditCard*  
static method), 668  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.datetime.DateTime*  
method), 671  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.datetime.DateTime*  
static method), 671  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.discrete.Discrete*  
method), 673  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.discrete.Discrete*  
static method), 674  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.gis.GIS*  
method), 676  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.gis.GIS*  
static method), 678  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.integer.Integer*  
method), 680  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.integer.Integer*  
static method), 681  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.lat\_long.LatLong*  
method), 689  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.lat\_long.LatLong*  
static method), 691  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.ordinal.Ordinal*  
method), 694  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.ordinal.Ordinal*  
static method), 694  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.string.String*  
method), 698  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.string.String*  
static method), 699  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.text.Text*  
method), 701  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.text.Text*  
static method), 701  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.zip\_code.ZipCode*  
method), 703  
**feature\_plot()** (*ads.feature\_engineering.feature\_type.zip\_code.ZipCode*  
static method), 704  
**feature\_select()** (*ads.feature\_engineering.accessor.dataframe\_accessor.DataFrameAccessor*  
method), 639, 640  
**feature\_select()** (*ads.feature\_engineering.accessor.mixin.eda\_mixin.EDA\_Mixin*  
method), 648  
**feature\_select()** (*ads.feature\_engineering.accessor.mixin.eda\_mixin.SeriesMixin*  
method), 650  
**feature\_select()** (*ads.feature\_engineering.feature\_type.address.Address*  
method), 654  
**feature\_select()** (*ads.feature\_engineering.feature\_type.address.Address*  
static method), 655

`feature_stat()` (`ads.feature_engineering.feature_type.book.Book` static method), 657  
`feature_stat()` (`ads.feature_engineering.feature_type.book.Book` static method), 658  
`feature_stat()` (`ads.feature_engineering.feature_type.category.Category` static method), 660  
`feature_stat()` (`ads.feature_engineering.feature_type.category.Category` static method), 661  
`feature_stat()` (`ads.feature_engineering.feature_type.confusion.Confusion` static method), 662  
`feature_stat()` (`ads.feature_engineering.feature_type.confusion.Confusion` static method), 663  
`feature_stat()` (`ads.feature_engineering.feature_type.confusion.Confusion` static method), 664  
`feature_stat()` (`ads.feature_engineering.feature_type.confusion.Confusion` static method), 665  
`feature_stat()` (`ads.feature_engineering.feature_type.create.Create` static method), 667  
`feature_stat()` (`ads.feature_engineering.feature_type.create.Create` static method), 669  
`feature_stat()` (`ads.feature_engineering.feature_type.data.Data` static method), 671  
`feature_stat()` (`ads.feature_engineering.feature_type.data.Data` static method), 672  
`feature_stat()` (`ads.feature_engineering.feature_type.discrete.Discrete` static method), 673  
`feature_stat()` (`ads.feature_engineering.feature_type.discrete.Discrete` static method), 674  
`feature_stat()` (`ads.feature_engineering.feature_type.gis.GIS` static method), 676  
`feature_stat()` (`ads.feature_engineering.feature_type.gis.GIS` static method), 678  
`feature_stat()` (`ads.feature_engineering.feature_type.integer.Integer` static method), 680  
`feature_stat()` (`ads.feature_engineering.feature_type.integer.Integer` static method), 681  
`feature_stat()` (`ads.feature_engineering.feature_type.ip_address.IPAddress` static method), 682  
`feature_stat()` (`ads.feature_engineering.feature_type.ip_address.IPAddress` static method), 683  
`feature_stat()` (`ads.feature_engineering.feature_type.ip_address.IPAddress` static method), 684  
`feature_stat()` (`ads.feature_engineering.feature_type.ip_address_v4.IPv4Address` static method), 685  
`feature_stat()` (`ads.feature_engineering.feature_type.ip_address_v4.IPv4Address` static method), 687  
`feature_stat()` (`ads.feature_engineering.feature_type.ip_address_v6.IPv6Address` static method), 688  
`feature_stat()` (`ads.feature_engineering.feature_type.lat_long.LatLong` static method), 689  
`feature_stat()` (`ads.feature_engineering.feature_type.lat_long.LatLong` static method), 691  
`feature_stat()` (`ads.feature_engineering.feature_type.ordinal.Ordinal` static method), 694  
`feature_stat()` (`ads.feature_engineering.feature_type.ordinal.Ordinal` static method), 695  
`feature_stat()` (`ads.feature_engineering.feature_type.phone_number.PhoneNumber` static method), 696  
`feature_stat()` (`ads.feature_engineering.feature_type.phone_number.PhoneNumber` static method), 697  
`feature_stat()` (`ads.feature_engineering.feature_type.string.String` static method), 698  
`feature_stat()` (`ads.feature_engineering.feature_type.string.String` static method), 699  
`feature_stat()` (`ads.feature_engineering.feature_type.zip_code.ZipCode` static method), 703  
`feature_stat()` (`ads.feature_engineering.feature_type.zip_code.ZipCode` static method), 704  
`feature_type()` (`ads.feature_engineering.accessor.dataframe_accessor.ADSDataset` attribute), 639  
`feature_type()` (`ads.feature_engineering.accessor.dataframe_accessor.ADSDataset` property), 640  
`feature_type()` (`ads.feature_engineering.accessor.series_accessor.ADSSeries` attribute), 644  
`feature_type()` (`ads.feature_engineering.accessor.series_accessor.ADSSeries` property), 644  
`feature_type_description` (`ads.feature_engineering.accessor.dataframe_accessor.ADSDataset` attribute), 639  
`feature_type_description` (`ads.feature_engineering.accessor.dataframe_accessor.ADSDataset` property), 640  
`feature_type_description` (`ads.feature_engineering.accessor.series_accessor.ADSSeries` attribute), 644  
`feature_type_description` (`ads.feature_engineering.accessor.series_accessor.ADSSeries` property), 645  
`feature_type_object()` (`ads.feature_engineering.feature_type_manager.FeatureTypeManager` static method), 636  
`feature_type_object()` (`ads.feature_engineering.feature_type_manager.FeatureTypeManager` static method), 635  
`feature_type_register()` (`ads.feature_engineering.feature_type_manager.FeatureTypeManager` static method), 636  
`feature_type_registered()` (`ads.feature_engineering.feature_type_manager.FeatureTypeManager` static method), 635  
`feature_type_reset()` (`ads.feature_engineering.feature_type_manager.FeatureTypeManager` static method), 635

- class method), 636
- feature\_type\_reset() (ads.feature\_engineering.feature\_type\_manager.FeatureTypeManager method), 635
- feature\_type\_unregister() (ads.feature\_engineering.feature\_type\_manager.FeatureTypeManager class method), 637
- feature\_type\_unregister() (ads.feature\_engineering.feature\_type\_manager.FeatureTypeManager method), 635
- FeatureBaseType (class in ads.feature\_engineering.feature\_type.base), 656
- FeatureBaseTypeMeta (class in ads.feature\_engineering.feature\_type.base), 656
- FeatureEngineeringTransformer (class in ads.dataset.feature\_engineering\_transformer), 615
- FeatureImportance (class in ads.dataset.feature\_selection), 615
- FeatureType (class in ads.feature\_engineering.feature\_type.base), 656
- FeatureTypeManager (class in ads.feature\_engineering.feature\_type\_manager), 634
- FeatureValidator (class in ads.feature\_engineering.feature\_type.handler.feature\_validator), 705
- FeatureValidatorMethod (class in ads.feature\_engineering.feature\_type.handler.feature\_validator), 708
- FeatureWarning (class in ads.feature\_engineering.feature\_type.handler.feature\_warning), 711
- fetch\_log() (ads.dataflow.dataflow.DataFlowRun method), 582
- FILE\_METADATA (ads.text\_dataset.options.Options attribute), 874
- FILE\_NAME (ads.text\_dataset.options.Options attribute), 874
- FileOption (class in ads.text\_dataset.options), 874
- FileOverwriteError, 529
- FileProcessor (class in ads.text\_dataset.extractor), 872
- FileProcessorFactory (class in ads.text\_dataset.extractor), 873
- filesystem() (ads.dataset.dataset\_browser.DatasetBrowser static method), 602
- filter() (ads.catalog.model.ModelSummaryList method), 500
- filter() (ads.catalog.notebook.NotebookSummaryList method), 503
- filter() (ads.catalog.project.ProjectSummaryList method), 506
- filter() (ads.catalog.summary.SummaryList method), 506
- filter() (ads.dataflow.dataflowssummary.SummaryList method), 584
- filter\_list() (ads.dataset.dataset\_browser.DatasetBrowser method), 602
- find\_not\_none() (in module ads.common.utils), 532
- fit() (ads.automl.provider.AutoMLFeatureSelection method), 488
- fit() (ads.automl.provider.AutoMLPreprocessingTransformer method), 489
- fit() (ads.automl.provider.BaselineModel method), 491
- fit() (ads.dataset.dataframe\_transformer.DataFrameTransformer method), 590
- fit() (ads.dataset.feature\_engineering\_transformer.FeatureEngineeringTransformer method), 615
- fit() (ads.dataset.label\_encoder.DataFrameLabelEncoder method), 619
- fit() (ads.dataset.recommendation\_transformer.RecommendationTransformer method), 621
- fit\_transform() (ads.dataset.feature\_engineering\_transformer.FeatureEngineeringTransformer method), 615
- fit\_transform() (ads.dataset.recommendation\_transformer.RecommendationTransformer method), 621
- fix\_column\_names() (in module ads.dataset.helper), 617
- flatten() (in module ads.common.utils), 532
- flatten\_corr\_matrix() (ads.dataset.correlation\_plot.BokehHeatMap method), 588
- folder\_size() (in module ads.common.utils), 532
- font\_sz (ads.evaluations.evaluation\_plot.EvaluationPlot attribute), 623, 624
- ForecastingDataset (class in ads.dataset.forecasting\_dataset), 616
- format (ads.dataset.helper.ElaboratedPath property), 616
- format() (ads.text\_dataset.dataset.TextDatasetFactory static method), 871
- Formats (class in ads.data\_labeling.constants), 544
- framework (ads.model.extractor.automl\_extractor.AutoMLExtractor property), 791
- framework (ads.model.extractor.keras\_extractor.KerasExtractor property), 797
- framework (ads.model.extractor.lightgbm\_extractor.LightgbmExtractor property), 793
- framework (ads.model.extractor.pytorch\_extractor.PyTorchExtractor property), 799
- framework (ads.model.extractor.sklearn\_extractor.SklearnExtractor property), 796
- framework (ads.model.extractor.tensorflow\_extractor.TensorflowExtractor property), 798



`framework` (`ads.model.extractor.xgboost_extractor.XgboostExtractor` class method), 790, 846  
     property), 792  
`framework` (`ads.model.framework.automl_model.AutoMLModel` static method), 518  
     attribute), 814  
`framework` (`ads.model.framework.lightgbm_model.LightGBMModel` class method), 562  
     attribute), 818  
`framework` (`ads.model.framework.pytorch_model.PyTorchModel` method), 560  
     attribute), 823  
`framework` (`ads.model.framework.sklearn_model.SklearnModel` class method), 566  
     attribute), 828  
`framework` (`ads.model.framework.tensorflow_model.TensorFlowModel` class method), 568  
     attribute), 834  
`framework` (`ads.model.framework.xgboost_model.XGBoostModel` class method), 745  
     attribute), 839  
`framework` (`ads.model.generic_model.GenericModel` attribute), 768  
`framework` (`ads.model.extractor.lightgbm_extractor.LightgbmExtractor` class method), 773  
     method), 793  
`framework` (`ads.model.extractor.model_info_extractor.ModelInfoExtractor` class method), 770  
     method), 794  
`framework` (`ads.model.extractor.pytorch_extractor.PyTorchExtractor` class method), 715  
     method), 799  
`framework` (`ads.model.extractor.sklearn_extractor.SklearnExtractor` class method), 816  
     method), 796  
`framework` (`ads.model.extractor.tensorflow_extractor.TensorFlowExtractor` class method), 820  
     method), 798  
`framework` (`ads.model.extractor.xgboost_extractor.XgboostExtractor` class method), 820  
     method), 792  
`FrameworkSpecificModel` (class in `ads.model.framework.pytorch_model.PyTorchModel`), 825  
     `ads.model.generic_model`), 765  
`from_dataflow_job` (`ads.jobs.ads_job.Job` static method), 732  
`from_dataframe` (`ads.dataset.factory.DatasetFactory` static method), 612  
`from_datascience_job` (`ads.jobs.ads_job.Job` static method), 732  
`from_dict` (`ads.data_labeling.visualizer.image_visualizer.RenderOptions` class method), 571  
`from_dict` (`ads.data_labeling.visualizer.text_visualizer.RenderOptions` class method), 573  
`from_dict` (`ads.jobs.ads_job.Job` class method), 732  
`from_dict` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` class method), 774  
     class method), 745  
`from_DLS` (`ads.data_labeling.reader.dataset_reader.LabeledDatasetReader` class method), 562  
`from_DLS` (`ads.data_labeling.reader.metadata_reader.MetadataReader` class method), 566  
`from_DLS` (`ads.data_labeling.reader.record_reader.RecordReader` class method), 567  
`from_dls_dataset` (`ads.data_labeling.metadata.Metadata` class method), 547  
`from_dsc_job` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` class method), 756  
`from_env` (`ads.model.runtime.runtime_info.RuntimeInfo` class method), 757  
`from_estimator` (`ads.common.model.ADSModel` class method), 770  
`from_export` (`ads.data_labeling.reader.dataset_reader.LabeledDatasetReader` class method), 562  
`from_export_file` (`ads.data_labeling.reader.metadata_reader.MetadataReader` class method), 566  
`from_export_file` (`ads.data_labeling.reader.record_reader.RecordReader` class method), 567  
`from_id` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` class method), 774  
`from_id` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` class method), 757  
`from_id` (`ads.model.generic_model.GenericModel` class method), 770  
`from_id` (`ads.model.generic_model.GenericModel` class method), 770  
`from_json` (`ads.hpo.distributions.DistributionEncoder` class method), 715  
`from_model_artifact` (`ads.model.framework.automl_model.AutoMLModel` class method), 816  
`from_model_artifact` (`ads.model.framework.lightgbm_model.LightGBMModel` class method), 562  
`from_model_artifact` (`ads.model.framework.pytorch_model.PyTorchModel` class method), 825  
`from_model_artifact` (`ads.model.framework.sklearn_model.SklearnModel` class method), 830  
`from_model_artifact` (`ads.model.framework.tensorflow_model.TensorFlowModel` class method), 835  
`from_model_artifact` (`ads.model.framework.xgboost_model.XGBoostModel` class method), 840  
`from_model_artifact` (`ads.model.generic_model.GenericModel` class method), 774  
`from_model_artifact` (`ads.model.generic_model.GenericModel` class method), 770  
`from_model_catalog` (`ads.model.framework.automl_model.AutoMLModel` class method), 816  
`from_model_catalog` (`ads.model.framework.lightgbm_model.LightGBMModel` class method), 820  
`from_model_catalog` (`ads.model.framework.pytorch_model.PyTorchModel` class method), 825

[from\\_model\\_catalog\(\)](#) (*ads.model.framework.sklearn\_model.SklearnModel* class method), 830  
[from\\_model\\_catalog\(\)](#) (*ads.model.framework.tensorflow\_model.TensorFlowModel* class method), 835  
[from\\_model\\_catalog\(\)](#) (*ads.model.framework.xgboost\_model.XGBoostModel* class method), 840  
[from\\_model\\_catalog\(\)](#) (*ads.model.generic\_model.GenericModel* class method), 775  
[from\\_model\\_catalog\(\)](#) (*ads.model.generic\_model.GenericModel* method), 770  
[from\\_model\\_deployment\(\)](#) (*ads.model.generic\_model.GenericModel* class method), 776  
[from\\_model\\_deployment\(\)](#) (*ads.model.generic\_model.GenericModel* method), 770  
[from\\_ocid\(\)](#) (*ads.jobs.builders.infrastructure.dsc\_job.DSCJob* class method), 754  
[from\\_path\(\)](#) (*ads.model.runtime.env\_info.EnvInfo* class method), 843  
[from\\_slug\(\)](#) (*ads.model.runtime.env\_info.EnvInfo* class method), 843  
[from\\_spacy\(\)](#) (*ads.data\_labeling.ner.NERItem* class method), 548  
[from\\_uri\(\)](#) (*ads.model.artifact.ModelArtifact* class method), 764  
[from\\_yolo\(\)](#) (*ads.data\_labeling.boundingbox.BoundingBox* class method), 542

**G**

[generate\\_fn\\_artifacts\(\)](#) (in module *ads.common.function.fn\_util*), 529  
[generate\\_heatmap\(\)](#) (*ads.dataset.correlation\_plot.BokehHeatMap* method), 588  
[generate\\_initial\\_types\(\)](#) (*ads.model.framework.lightgbm\_model.LightGBMModel* method), 822  
[generate\\_initial\\_types\(\)](#) (*ads.model.framework.sklearn\_model.SklearnModel* method), 832  
[generate\\_initial\\_types\(\)](#) (*ads.model.framework.xgboost\_model.XGBoostModel* method), 842  
[generate\\_requirement\\_file\(\)](#) (in module *ads.common.utils*), 533  
[generate\\_sample\(\)](#) (in module *ads.dataset.helper*), 617  
[generate\\_target\\_heatmap\(\)](#) (*ads.dataset.correlation\_plot.BokehHeatMap* method), 588  
[get\\_app\(\)](#) (*ads.dataflow.dataflow.DataFlow* method), 576  
[get\\_base\\_modules\(\)](#) (in module *ads.common.utils*), 533  
[get\\_bootstrap\\_styles\(\)](#) (in module *ads.common.utils*), 533  
[get\\_compute\\_accelerator\\_ncores\(\)](#) (in module *ads.common.utils*), 533  
[get\\_cpu\\_count\(\)](#) (in module *ads.common.utils*), 533  
[get\\_data\\_serializer\(\)](#) (*ads.model.framework.pytorch\_model.PyTorchModel* method), 826  
[get\\_data\\_serializer\(\)](#) (*ads.model.framework.tensorflow\_model.TensorFlowModel* method), 837  
[get\\_data\\_serializer\(\)](#) (*ads.model.generic\_model.GenericModel* method), 777  
[get\\_dataframe\\_styles\(\)](#) (in module *ads.common.utils*), 533  
[get\\_distribution\(\)](#) (*ads.hpo.distributions.Distribution* method), 714  
[get\\_dtype\(\)](#) (in module *ads.dataset.helper*), 617  
[get\\_feature\\_type\(\)](#) (in module *ads.dataset.helper*), 617  
[get\\_files\(\)](#) (in module *ads.common.utils*), 533  
[get\\_fill\\_val\(\)](#) (in module *ads.dataset.helper*), 617  
[get\\_format\\_reader\(\)](#) (in module *ads.dataset.factory*), 615  
[get\\_function\\_config\(\)](#) (in module *ads.common.function.fn\_util*), 529  
[get\\_init\\_types\(\)](#) (*ads.common.model.ADSModel* static method), 519  
[get\\_legend\\_labels\(\)](#) (*ads.evaluations.evaluation\_plot.EvaluationPlot* class method), 624  
[get\\_legend\\_labels\(\)](#) (*ads.evaluations.evaluation\_plot.EvaluationPlot* method), 623  
[get\\_metadata\(\)](#) (*ads.text\_dataset.backends.Base* method), 865  
[get\\_metadata\(\)](#) (*ads.text\_dataset.backends.PDFPlumber* method), 866  
[get\\_metadata\(\)](#) (*ads.text\_dataset.backends.Tika* method), 867  
[get\\_metadata\(\)](#) (*ads.text\_dataset.extractor.FileProcessor* method), 872  
[get\\_metrics\(\)](#) (*ads.evaluations.statistical\_metrics.ModelEvaluator*

method), 631, 632

get\_ml\_task\_type() (in module *ads.automl.driver*), 488

get\_model() (*ads.catalog.model.ModelCatalog* method), 496, 498

get\_model\_deployment() (*ads.model.deployment.model\_deployer.ModelDeployer* method), 801, 803

get\_model\_deployment\_state() (*ads.model.deployment.model\_deployer.ModelDeployer* method), 801, 803

get\_notebook\_session() (*ads.catalog.notebook.NotebookCatalog* method), 502

get\_oci\_config() (in module *ads.common.utils*), 533

get\_processor() (*ads.text\_dataset.extractor.FileProcessor* static method), 873

get\_progress\_bar() (in module *ads.common.utils*), 533

get\_project() (*ads.catalog.project.ProjectCatalog* method), 504

get\_random\_name\_for\_resource() (in module *ads.common.utils*), 534

get\_recommendations() (*ads.dataset.dataset\_with\_target.ADSDatasetWithTarget* method), 606

get\_repository() (in module *ads.database.connection*), 575

get\_run() (*ads.dataflow.dataflow.DataFlowApp* method), 579

get\_secret() (*ads.vault.vault.Vault* method), 875

get\_service\_packs() (in module *ads.model.runtime.utils*), 847

get\_signer() (in module *ads.common.auth*), 514

get\_sqlalchemy\_engine() (in module *ads.common.utils*), 534

get\_status() (*ads.hpo.search\_cv.ADSTuner* method), 718

get\_transformed\_dataset() (*ads.dataset.dataset\_with\_target.ADSDatasetWithTarget* method), 607

get\_transformer\_pipeline() (*ads.automl.provider.AutoMLProvider* method), 489

get\_transformer\_pipeline() (*ads.automl.provider.BaselineAutoMLProvider* method), 490

get\_transformer\_pipeline() (*ads.automl.provider.OracleAutoMLProvider* method), 492

get\_value() (in module *ads.common.utils*), 534

getLogger() (in module *ads*), 877

GIS (class in *ads.feature\_engineering.feature\_type.gis*), 676

GitHub() (*ads.dataset.dataset\_browser.DatasetBrowser* static method), 602

GitHubDatasets (class in *ads.dataset.dataset\_browser*), 603

GitPythonRuntime (class in *ads.jobs.builders.runtimes.python\_runtime*), 739

## H

halt() (*ads.hpo.search\_cv.ADSTuner* method), 719

HALTED (*ads.hpo.search\_cv.State* attribute), 728

handle() (*ads.text\_dataset.options.FileOption* method), 874

handle() (*ads.text\_dataset.options.MetadataOption* method), 874

handle() (*ads.text\_dataset.options.OptionHandler* method), 874

has\_kerberos\_ticket() (in module *ads.bds.auth*), 540

hdfs\_host (*ads.secrets.big\_data\_service.BDSSecret* attribute), 859, 860

hdfs\_host (*ads.secrets.big\_data\_service.BDSSecretKeeper* attribute), 860

hdfs\_port (*ads.secrets.big\_data\_service.BDSSecret* attribute), 859, 860

hdfs\_port (*ads.secrets.big\_data\_service.BDSSecretKeeper* attribute), 861

head() (*ads.dataflow.dataflow.DataFlowLog* method), 580

hello() (in module *ads*), 877

help() (*ads.feature\_engineering.accessor.dataframe\_accessor.ADSDatasetAccessor* method), 639

help() (*ads.feature\_engineering.accessor.mixin.feature\_types\_mixin.ADSDatasetAccessor* method), 651

help() (*ads.feature\_engineering.accessor.series\_accessor.ADSSeriesAccessor* method), 643

high\_cardinality\_handler() (in module *ads.feature\_engineering.feature\_type.handler.warnings*), 713

highlight\_text() (in module *ads.common.utils*), 534

hive\_host (*ads.secrets.big\_data\_service.BDSSecret* attribute), 859, 860

hive\_host (*ads.secrets.big\_data\_service.BDSSecretKeeper* attribute), 861

hive\_port (*ads.secrets.big\_data\_service.BDSSecret* attribute), 859, 860

hive\_port (*ads.secrets.big\_data\_service.BDSSecretKeeper* attribute), 861

horizontal\_scrollable\_div() (in module *ads.common.utils*), 534

host (*ads.secrets.mysqlpdb.MySQLDBSecret* attribute), 855

host (*ads.secrets.oracledb.OracleDBSecret* attribute), 857

human\_size() (in module *ads.common.utils*), 535

`hyperparameter` (`ads.model.extractor.automl_extractor.AutoMLExtractor` attribute), 791  
`hyperparameter` (`ads.model.extractor.keras_extractor.KerasExtractor` attribute), 797  
`hyperparameter` (`ads.model.extractor.lightgbm_extractor.LightGBMExtractor` attribute), 794  
`hyperparameter` (`ads.model.extractor.pytorch_extractor.PyTorchExtractor` attribute), 799  
`hyperparameter` (`ads.model.extractor.sklearn_extractor.SklearnExtractor` attribute), 796  
`hyperparameter` (`ads.model.extractor.tensorflow_extractor.TensorFlowExtractor` attribute), 798  
`hyperparameter` (`ads.model.extractor.xgboost_extractor.XgboostExtractor` attribute), 792  
`hyperparameter` (`ads.model.framework.automl_model.AutoMLModel` attribute), 814  
`hyperparameter` (`ads.model.framework.lightgbm_model.LightGBMModel` attribute), 818  
`hyperparameter` (`ads.model.framework.pytorch_model.PyTorchModel` attribute), 823  
`hyperparameter` (`ads.model.framework.sklearn_model.SklearnModel` attribute), 829  
`hyperparameter` (`ads.model.framework.tensorflow_model.TensorFlowModel` attribute), 834  
`hyperparameter` (`ads.model.framework.xgboost_model.XGBoostModel` attribute), 839  
`hyperparameter` (`ads.model.generic_model.GenericModel` attribute), 768  
`hyperparameter`() (`ads.model.extractor.lightgbm_extractor.LightGBMExtractor` method), 793  
`hyperparameter`() (`ads.model.extractor.model_info_extractor.ModelInfoExtractor` method), 794, 795  
`hyperparameter`() (`ads.model.extractor.pytorch_extractor.PyTorchExtractor` method), 799  
`hyperparameter`() (`ads.model.extractor.sklearn_extractor.SklearnExtractor` method), 796  
`hyperparameter`() (`ads.model.extractor.tensorflow_extractor.TensorFlowExtractor` method), 798  
`hyperparameter`() (`ads.model.extractor.xgboost_extractor.XgboostExtractor` method), 792  
  
**I**  
`id` (`ads.jobs.ads_job.Job` property), 732  
`identify_issue_network`() (`ads.common.card_identifier.card_identify` method), 507  
**IMAGE** (`ads.data_labeling.constants.DatasetType` attribute), 544  
**ImageLabeledDataFormatter** (class in `ads.data_labeling.visualizer.image_visualizer`), 570  
**IMAGEOBJECTSELECTION** (`ads.data_labeling.parser.export_record_parser.EntityType` attribute), 554  
  
`image_visualizer` (`ads.data_labeling.visualizer.image_visualizer.LabeledImageItem` attribute), 570, 571  
`importwallet`() (in module `ads.database.connection`), 575  
`infer_fn` (`ads.dataset.factory.DatasetFactory` class method), 612  
`inference_conda_env` (`ads.model.model_properties.ModelProperties` attribute), 789  
`inference_conda_env` (`ads.model.runtime.model_deployment_details.ModelDeploymentDetails` attribute), 845  
`inference_env_path` (`ads.model.runtime.env_info.InferenceEnvInfo` attribute), 844  
`inference_env_slug` (`ads.model.runtime.env_info.InferenceEnvInfo` attribute), 844  
`inference_env_type` (`ads.model.runtime.env_info.InferenceEnvInfo` attribute), 844  
`inference_python_version` (`ads.model.model_properties.ModelProperties` attribute), 789  
`inference_python_version` (`ads.model.runtime.env_info.InferenceEnvInfo` attribute), 844  
`inference_env_info` (class in `ads.model.runtime.env_info`), 844  
`info`() (`ads.data_labeling.interface.reader.Reader` attribute), 541  
`info`() (`ads.data_labeling.reader.dataset_reader.DLSDatasetReader` attribute), 559  
`info`() (`ads.data_labeling.reader.dataset_reader.ExportReader` attribute), 559, 560  
`info`() (`ads.data_labeling.reader.dataset_reader.LabeledDatasetReader` attribute), 560, 562  
`info`() (`ads.feature_engineering.accessor.dataframe_accessor.ADSDatasetReader` attribute), 794, 795  
`infrastructure` (`ads.jobs.ads_job.Job` property), 732  
`init_ccache_with_keytab`() (in module `ads.bds.auth`), 540  
`init_client`() (`ads.jobs.builders.infrastructure.dataflow.DataFlowApp` class method), 750  
`init_client`() (`ads.jobs.builders.infrastructure.dataflow.DataFlowRun` class method), 751  
`INITIATED` (`ads.hpo.search_cv.State` attribute), 728  
`inject_and_copy_kwargs`() (in module `ads.common.utils`), 535  
`INSTANCE_PRINCIPAL` (`ads.common.auth.AuthType` attribute), 509  
`instance_shapes`() (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` attribute), 757  
`InstancePrincipal` (class in `ads.common.auth`), 509



Integer (class in `ads.feature_engineering.feature_type.integer`), 719  
 680 is\_notebook() (in module `ads.common.utils`), 536  
 IntLogUniformDistribution (class in `ads.hpo.distributions`), 715 is\_resource\_principal\_mode() (in module `ads.common.utils`), 536  
 introspect() (`ads.model.framework.automl_model.AutoMLModel` method), 816 is\_training() (`ads.hpo.search_cv.ADSTuner` method), 720  
 introspect() (`ads.model.framework.lightgbm_model.LightGBMModel` method), 820 issubclass() (in module `ads.common.utils`), 536  
 introspect() (`ads.model.framework.pytorch_model.PyTorchModel` method), 720 is\_terminated() (`ads.hpo.search_cv.ADSTuner` method), 825  
 introspect() (`ads.model.framework.sklearn_model.SklearnModel` method), 830 is\_test() (in module `ads.common.utils`), 536  
 introspect() (`ads.model.framework.tensorflow_model.TensorFlowModel` method), 835 is\_text\_data() (in module `ads.dataset.helper`), 617  
 introspect() (`ads.model.framework.xgboost_model.XGBoostModel` method), 840 is\_type\_registered() (in module `ads.feature_engineering.feature_type_manager.FeatureTypeManager`), 637  
 introspect() (`ads.model.generic_model.GenericModel` method), 770, 777 items (`ads.data_labeling.ner.NERItems` attribute), 549  
 Introspectable (class in `ads.common.model_introspect`), 524 **J**  
 IntrospectionNotPassed, 524 job (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJobRun` property), 762  
 IntUniformDistribution (class in `ads.hpo.distributions`), 716 Job (class in `ads.jobs.ads_job`), 729  
 InvalidFeatureType, 633 job\_id (`ads.jobs.builders.infrastructure.dataflow.DataFlow` property), 745  
 InvalidStateTransition, 728 job\_id (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` property), 757  
 ip (`ads.feature_engineering.adsstring.common_regex_mixin.CommonRegexMixin` property), 652 job\_infrastructure\_type (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` property), 757  
 IPAddress (class in `ads.feature_engineering.feature_type.ip_address`), 682  
 IPAddressV4 (class in `ads.feature_engineering.feature_type.ip_address_v4`), 684 job\_type (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` property), 757  
 IPAddressV6 (class in `ads.feature_engineering.feature_type.ip_address_v6`), 686 JsonConverter (class in `ads.common.utils`), 529  
 IpythonProgressBar (class in `ads.dataset.progress`), 620 in JsonlReader (class in `ads.data_labeling.reader.jsonl_reader`), 563  
 is\_balanced() (`ads.dataset.target.TargetVariable` method), 622 **K**  
 is\_classifier (`ads.evaluations.evaluator.ADSEvaluator` attribute), 625 KerasExtractor (class in `ads.model.extractor.keras_extractor`), 797  
 is\_classifier() (`ads.common.model.ADSTuner` method), 519 kerb5\_content (`ads.secrets.big_data_service.BDSSecret` attribute), 859, 860  
 is\_completed() (`ads.hpo.search_cv.ADSTuner` method), 719 kerb5\_content (`ads.secrets.big_data_service.BDSSecretKeeper` attribute), 861  
 is\_data\_too\_wide() (in module `ads.common.utils`), 535 kerb5\_path (`ads.secrets.big_data_service.BDSSecret` attribute), 859, 860  
 is\_debug\_mode() (in module `ads.common.utils`), 535 kerb5\_path (`ads.secrets.big_data_service.BDSSecretKeeper` attribute), 861  
 is\_documentation\_mode() (in module `ads.common.utils`), 536 key (`ads.common.model_introspect.PrintItem` attribute), 525  
 is\_either\_numerical\_or\_string\_dataframe() (`ads.model.framework.sklearn_model.SklearnModel` static method), 832 key\_id (`ads.secrets.big_data_service.BDSSecretKeeper` attribute), 862  
 is\_halted() (`ads.hpo.search_cv.ADSTuner` method), keytab\_content (`ads.secrets.big_data_service.BDSSecret` attribute), 860  
 keytab\_content (`ads.secrets.big_data_service.BDSSecretKeeper` attribute), 861

keytab\_path (*ads.secrets.big\_data\_service.BDSecretKeeper* attribute), 859, 860  
 keytab\_path (*ads.secrets.big\_data\_service.BDSecretKeeper* attribute), 861  
 kind (*ads.jobs.ads\_job.Job* property), 733  
 KRB5KinitError, 539  
 krbcontext() (in module *ads.bds.auth*), 540  
 kwargs (*ads.secrets.big\_data\_service.BDSecretKeeper* attribute), 861

## L

label (*ads.data\_labeling.ner.NERItem* attribute), 548  
 LabeledDatasetReader (class in *ads.data\_labeling.reader.dataset\_reader*), 560  
 LabeledImageItem (class in *ads.data\_labeling.visualizer.image\_visualizer*), 570  
 LabeledTextItem (class in *ads.data\_labeling.visualizer.text\_visualizer*), 572  
 labels (*ads.data\_labeling.boundingbox.BoundingBoxItem* attribute), 542, 543  
 labels (*ads.data\_labeling.metadata.Metadata* attribute), 546, 547  
 LABS (*ads.common.decorator.runtime\_dependency.OptionalDependency* attribute), 522  
 LatLong (class in *ads.feature\_engineering.feature\_type.lat\_long*), 689  
 legend\_labels (*ads.evaluations.evaluator.ADSEvaluator* attribute), 625  
 length (*ads.data\_labeling.ner.NERItem* attribute), 548  
 less\_is\_more (*ads.evaluations.evaluator.ADSEvaluator.EvaluationMetric* attribute), 627  
 LightgbmExtractor (class in *ads.model.extractor.lightgbm\_extractor*), 793  
 LightGBMModel (class in *ads.model.framework.lightgbm\_model*), 818  
 link (*ads.feature\_engineering.adsstring.common\_regex\_mixin.CommonRegexMixin* property), 652  
 list() (*ads.dataset.dataset\_browser.DatasetBrowser* static method), 602  
 list() (*ads.dataset.dataset\_browser.GitHubDatasets* method), 603  
 list() (*ads.dataset.dataset\_browser.LocalFilesystemDatasets* method), 603  
 list() (*ads.dataset.dataset\_browser.SeabornDatasets* method), 604  
 list() (*ads.dataset.dataset\_browser.SklearnDatasets* method), 604  
 list() (*ads.dataset.dataset\_browser.WebDatasets* method), 605  
 list\_apps() (*ads.dataflow.dataflow.DataFlow* method), 577  
 list\_dataset() (*ads.data\_labeling.data\_labeling\_service.DataLabelingService* method), 546  
 list\_deployments() (*ads.model.deployment.model\_deployer.ModelDeployer* method), 801, 803  
 list\_jobs() (*ads.jobs.builders.infrastructure.dataflow.DataFlow* class method), 745  
 list\_jobs() (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob* class method), 757  
 list\_model\_deployment() (*ads.catalog.model.ModelCatalog* method), 496, 498  
 list\_models() (*ads.catalog.model.ModelCatalog* method), 496, 499  
 list\_notebook\_session() (*ads.catalog.notebook.NotebookCatalog* method), 502  
 list\_projects() (*ads.catalog.project.ProjectCatalog* method), 504  
 list\_runs() (*ads.dataflow.dataflow.DataFlowApp* method), 579  
 list\_snapshots() (*ads.dataset.factory.DatasetFactory* static method), 612  
 list\_workflow\_logs() (*ads.model.deployment.model\_deployment.ModelDeployment* method), 806, 807  
 load() (*ads.data\_labeling.interface.loader.Loader* method), 541  
 load\_app() (*ads.dataflow.dataflow.DataFlow* method), 577  
 load\_dataset() (in module *ads.dataset.factory*), 615  
 load\_model() (*ads.catalog.model.Model* class method), 495  
 load\_model() (*ads.catalog.model.Model* method), 494  
 load\_properties\_from\_env() (*ads.jobs.builders.infrastructure.dsc\_job.DSCJob* method), 754  
 load\_secret() (*ads.secrets.secrets.SecretKeeper* class method), 849  
 Loader (class in *ads.data\_labeling.interface.loader*), 541  
 local\_dir (*ads.dataflow.dataflow.DataFlowLog* property), 581  
 local\_dir (*ads.dataflow.dataflow.DataFlowRun* property), 582  
 local\_dir (*ads.dataflow.dataflow.RunObserver* property), 583  
 local\_path (*ads.dataflow.dataflow.DataFlowLog* property), 581  
 LocalFilesystemDatasets (class in *ads.dataset.dataset\_browser*), 603  
 log\_group\_id (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob* property), 757  
 log\_group\_id (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun* property), 757

property), 762

log\_id(*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob* property), 758

log\_id(*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun* property), 762

LOG\_OUTPUTS (*ads.dataflow.dataflow.DataFlowRun* attribute), 582

log\_stderr (*ads.dataflow.dataflow.DataFlowRun* property), 582

log\_stdout (*ads.dataflow.dataflow.DataFlowRun* property), 582

logging (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun* property), 762

LogNotConfiguredError, 804

logs (*ads.jobs.builders.infrastructure.dataflow.DataFlowRun* property), 751

logs() (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun* method), 762

logs() (*ads.model.deployment.model\_deployment.ModelDeployment* method), 807

LogUniformDistribution (class in *ads.hpo.distributions*), 716

## M

map\_types() (in module *ads.dataset.helper*), 617

merge() (*ads.dataset.dataset.ADSDataset* method), 593

message (*ads.common.model\_introspect.PrintItem* attribute), 526

Metadata (class in *ads.data\_labeling.metadata*), 546

metadata\_all() (*ads.text\_dataset.dataset.DataLoader* method), 869

metadata\_custom(*ads.model.framework.automl\_model.AutoMLModel* attribute), 814

metadata\_custom(*ads.model.framework.lightgbm\_model.LightGBMModel* attribute), 818

metadata\_custom(*ads.model.framework.pytorch\_model.PyTorchModel* attribute), 824

metadata\_custom(*ads.model.framework.sklearn\_model.SklearnModel* attribute), 829

metadata\_custom(*ads.model.framework.tensorflow\_model.TensorFlowModel* attribute), 834

metadata\_custom(*ads.model.framework.xgboost\_model.XGBoostModel* attribute), 839

metadata\_custom(*ads.model.generic\_model.GenericModel* attribute), 768

metadata\_custom(*ads.model.generic\_model.GenericModel* property), 777

metadata\_provenance (*ads.model.framework.automl\_model.AutoMLModel* attribute), 814

metadata\_provenance (*ads.model.framework.lightgbm\_model.LightGBMModel* attribute), 818

metadata\_provenance (*ads.model.framework.pytorch\_model.PyTorchModel* attribute), 824

metadata\_provenance (*ads.model.framework.sklearn\_model.SklearnModel* attribute), 829

metadata\_provenance (*ads.model.framework.tensorflow\_model.TensorFlowModel* attribute), 834

metadata\_provenance (*ads.model.framework.xgboost\_model.XGBoostModel* attribute), 839

metadata\_provenance (*ads.model.generic\_model.GenericModel* attribute), 768

metadata\_provenance (*ads.model.generic\_model.GenericModel* property), 777

MetadataMixin (class in *ads.common.model\_metadata\_mixin*), 538

MetadataOption (class in *ads.text\_dataset.options*), 874

MetadataParser (class in *ads.data\_labeling.parser.export\_metadata\_parser*), 553

MetadataReader (class in *ads.data\_labeling.reader.metadata\_reader*), 565

METHOD (*ads.common.decorator.deprecate.TARGET\_TYPE* attribute), 523

metrics (*ads.evaluations.evaluator.ADSEvaluator* property), 629

metrics (*ads.evaluations.statistical\_metrics.ModelEvaluator* attribute), 631

metrics\_to\_show (*ads.evaluations.evaluator.ADSEvaluator* attribute), 625

missing\_values\_handler() (in module

`ads.feature_engineering.feature_type.handler.warnings)`, `attribute`), 790, 846  
 713 `model_deployment_id`  
`ml_task_types` (class in `ads.common.utils`), 536  
`model` (`ads.model.extractor.automl_extractor.AutoMLExtractor` `attribute`), 805  
`attribute`), 791 `model_deployment_id`  
`model` (`ads.model.extractor.keras_extractor.KerasExtractor` `attribute`), 797  
`attribute`), 797 (`ads.model.generic_model.GenericModel` `property`), 777  
`model` (`ads.model.extractor.lightgbm_extractor.LightgbmExtractor` `attribute`), 793  
`attribute`), 793 `model_file_name` (`ads.model.framework.automl_model.AutoMLModel` `attribute`), 815  
`model` (`ads.model.extractor.pytorch_extractor.PyTorchExtractor` `attribute`), 799  
`attribute`), 799 `model_file_name` (`ads.model.framework.lightgbm_model.LightGBMModel` `attribute`), 819  
`model` (`ads.model.extractor.sklearn_extractor.SklearnExtractor` `attribute`), 795  
`attribute`), 795 `model_file_name` (`ads.model.framework.pytorch_model.PyTorchModel` `attribute`), 824  
`model` (`ads.model.extractor.tensorflow_extractor.TensorflowExtractor` `attribute`), 798  
`attribute`), 798 `model_file_name` (`ads.model.framework.sklearn_model.SklearnModel` `attribute`), 829  
`model` (`ads.model.extractor.xgboost_extractor.XgboostExtractor` `attribute`), 792  
`attribute`), 792 `model_file_name` (`ads.model.framework.tensorflow_model.TensorFlowModel` `attribute`), 834  
`MODEL` (`ads.model.generic_model.DataScienceModelType` `attribute`), 765  
`attribute`), 765 `model_file_name` (`ads.model.framework.xgboost_model.XGBoostModel` `attribute`), 839  
`Model` (class in `ads.catalog.model`), 494  
`model_file_name` (`ads.model.generic_model.GenericModel` `attribute`), 769  
`model_artifact` (`ads.model.framework.automl_model.AutoMLModel` `attribute`), 814  
`attribute`), 814 `model_id` (`ads.model.deployment.model_deployment_properties.ModelDeploymentProperties` `attribute`), 809  
`model_artifact` (`ads.model.framework.lightgbm_model.LightGBMModel` `attribute`), 819  
`attribute`), 819 `model_id` (`ads.model.framework.automl_model.AutoMLModel` `attribute`), 815  
`model_artifact` (`ads.model.framework.pytorch_model.PyTorchModel` `attribute`), 824  
`attribute`), 824 `model_id` (`ads.model.framework.lightgbm_model.LightGBMModel` `attribute`), 819  
`model_artifact` (`ads.model.framework.sklearn_model.SklearnModel` `attribute`), 829  
`attribute`), 829 `model_id` (`ads.model.framework.pytorch_model.PyTorchModel` `attribute`), 824  
`model_artifact` (`ads.model.framework.tensorflow_model.TensorFlowModel` `attribute`), 834  
`attribute`), 834 `model_id` (`ads.model.framework.sklearn_model.SklearnModel` `attribute`), 829  
`model_artifact` (`ads.model.framework.xgboost_model.XGBoostModel` `attribute`), 839  
`attribute`), 839 `model_id` (`ads.model.framework.tensorflow_model.TensorFlowModel` `attribute`), 834  
`model_artifact` (`ads.model.generic_model.GenericModel` `attribute`), 769  
`attribute`), 769 `model_id` (`ads.model.framework.xgboost_model.XGBoostModel` `attribute`), 839  
`model_artifact_version`  
`(ads.model.runtime.runtime_info.RuntimeInfo` `attribute`), 790, 846  
`attribute`), 790, 846 `model_id` (`ads.model.generic_model.GenericModel` `attribute`), 769  
`model_deployment` (`ads.model.framework.automl_model.AutoMLModel` `attribute`), 815  
`attribute`), 815 `model_id` (`ads.model.generic_model.GenericModel` `property`), 777  
`model_deployment` (`ads.model.framework.lightgbm_model.LightGBMModel` `attribute`), 819  
`attribute`), 819 `model_id` (`ads.model.deployment.model_deployment_properties.ModelDeploymentProperties` `attribute`), 631  
`model_deployment` (`ads.model.framework.pytorch_model.PyTorchModel` `attribute`), 824  
`attribute`), 824 `model_id` (`ads.model.runtime.runtime_info.RuntimeInfo` `attribute`), 790, 846  
`model_deployment` (`ads.model.framework.sklearn_model.SklearnModel` `attribute`), 829  
`attribute`), 829 `model_id` (`ads.feature_engineering.accessor.dataframe_accessor.DataFrameAccessor` `method`), 641  
`model_deployment` (`ads.model.framework.tensorflow_model.TensorFlowModel` `attribute`), 834  
`attribute`), 834 `model_id` (`ads.model.deployment.model_deployment_properties.ModelDeploymentProperties` `attribute`), 810  
`model_deployment` (`ads.model.framework.xgboost_model.XGBoostModel` `attribute`), 839  
`attribute`), 839 `ModelArtifact` (class in `ads.model.artifact`), 763  
`ModelArtifactSizeError`, 496  
`MODEL_DEPLOYMENT` (`ads.model.generic_model.DataScienceModelType` `attribute`), 765  
`attribute`), 765 `ModelCatalog` (class in `ads.catalog.model`), 496  
`ModelDeployer` (class in `ads.model.deployment.model_deployer`), 801  
`model_deployment` (`ads.model.generic_model.GenericModel` `attribute`), 769  
`attribute`), 769  
`model_deployment` (`ads.model.runtime.runtime_info.RuntimeInfo` `attribute`), 790, 846  
`attribute`), 790, 846 `ModelDeployment` (class in `ads.model.deployment.model_deployment_properties.ModelDeploymentProperties`), 801  
`ModelDeployment` (class in `ads.model.deployment.model_deployment_properties.ModelDeploymentProperties`), 801



[ads.model.deployment.model\\_deployment](#)),  
[804](#)  
[ModelDeploymentDetails](#) (class in  
[ads.model.runtime.model\\_deployment\\_details](#)),  
[845](#)  
[ModelDeploymentLogType](#) (class in  
[ads.model.deployment.model\\_deployment](#)),  
[809](#)  
[ModelDeploymentProperties](#) (class in  
[ads.model.deployment.model\\_deployment\\_properties](#)),  
[809](#)  
[ModelEvaluator](#) (class in  
[ads.evaluations.statistical\\_metrics](#)), [631](#)  
[ModelInfoExtractor](#) (class in  
[ads.model.extractor.model\\_info\\_extractor](#)),  
[794](#)  
[ModelInfoExtractorFactory](#) (class in  
[ads.model.extractor.model\\_info\\_extractor\\_factory](#)),  
[790](#)  
[ModelIntrospect](#) (class in  
[ads.common.model\\_introspect](#)), [524](#)  
[ModelProperties](#) (class in  
[ads.model.model\\_properties](#)), [788](#)  
[ModelProvenanceDetails](#) (class in  
[ads.model.runtime.model\\_provenance\\_details](#)),  
[845](#)  
[models](#) ([ads.evaluations.evaluator.ADSEvaluator](#) at-  
[tribute](#)), [625](#)  
[ModelState](#) (class in [ads.model.generic\\_model](#)), [787](#)  
[ModelSummaryList](#) (class in [ads.catalog.model](#)), [500](#)  
[ModelWithActiveDeploymentError](#), [501](#)  
[module](#)  
[ads](#), [877](#)  
[ads.automl](#), [494](#)  
[ads.automl.driver](#), [487](#)  
[ads.automl.provider](#), [488](#)  
[ads.bds](#), [541](#)  
[ads.bds.auth](#), [539](#)  
[ads.catalog](#), [507](#)  
[ads.catalog.model](#), [494](#)  
[ads.catalog.notebook](#), [501](#)  
[ads.catalog.project](#), [503](#)  
[ads.catalog.summary](#), [506](#)  
[ads.common](#), [538](#)  
[ads.common.auth](#), [507](#)  
[ads.common.card\\_identifier](#), [507](#)  
[ads.common.data](#), [516](#)  
[ads.common.decorator.deprecate](#), [523](#)  
[ads.common.decorator.runtime\\_dependency](#),  
[521](#)  
[ads.common.function.fn\\_util](#), [529](#)  
[ads.common.model](#), [518](#)  
[ads.common.model\\_export\\_util](#), [526](#)  
[ads.common.model\\_introspect](#), [524](#)  
[ads.common.model\\_metadata](#), [521](#)  
[ads.common.model\\_metadata\\_mixin](#), [538](#)  
[ads.common.utils](#), [529](#)  
[ads.config](#), [876](#)  
[ads.data\\_labeling](#), [575](#)  
[ads.data\\_labeling.boundingBox](#), [542](#)  
[ads.data\\_labeling.constants](#), [544](#)  
[ads.data\\_labeling.data\\_labeling\\_service](#),  
[545](#)  
[ads.data\\_labeling.interface.loader](#), [541](#)  
[ads.data\\_labeling.interface.parser](#), [541](#)  
[ads.data\\_labeling.interface.reader](#), [541](#)  
[ads.data\\_labeling.metadata](#), [546](#)  
[ads.data\\_labeling.mixin.data\\_labeling](#),  
[550](#)  
[ads.data\\_labeling.ner](#), [548](#)  
[ads.data\\_labeling.parser.export\\_metadata\\_parser](#),  
[553](#)  
[ads.data\\_labeling.parser.export\\_record\\_parser](#),  
[553](#)  
[ads.data\\_labeling.reader.dataset\\_reader](#),  
[557](#)  
[ads.data\\_labeling.reader.jsonl\\_reader](#),  
[563](#)  
[ads.data\\_labeling.reader.metadata\\_reader](#),  
[564](#)  
[ads.data\\_labeling.reader.record\\_reader](#),  
[567](#)  
[ads.data\\_labeling.record](#), [549](#)  
[ads.data\\_labeling.visualizer.image\\_visualizer](#),  
[569](#)  
[ads.data\\_labeling.visualizer.text\\_visualizer](#),  
[572](#)  
[ads.database](#), [576](#)  
[ads.database.connection](#), [575](#)  
[ads.dataflow](#), [584](#)  
[ads.dataflow.dataflow](#), [576](#)  
[ads.dataflow.dataflowschema](#), [584](#)  
[ads.dataset](#), [623](#)  
[ads.dataset.classification\\_dataset](#), [584](#)  
[ads.dataset.correlation](#), [588](#)  
[ads.dataset.correlation\\_plot](#), [588](#)  
[ads.dataset.dataframe\\_transformer](#), [590](#)  
[ads.dataset.dataset](#), [591](#)  
[ads.dataset.dataset\\_browser](#), [602](#)  
[ads.dataset.dataset\\_with\\_target](#), [605](#)  
[ads.dataset.exception](#), [610](#)  
[ads.dataset.factory](#), [610](#)  
[ads.dataset.feature\\_engineering\\_transformer](#),  
[615](#)  
[ads.dataset.feature\\_selection](#), [615](#)  
[ads.dataset.forecasting\\_dataset](#), [616](#)  
[ads.dataset.helper](#), [616](#)  
[ads.dataset.label\\_encoder](#), [619](#)

`ads.dataset.pipeline`, 619  
`ads.dataset.plot`, 619  
`ads.dataset.progress`, 620  
`ads.dataset.recommendation`, 620  
`ads.dataset.recommendation_transformer`, 620  
`ads.dataset.regression_dataset`, 621  
`ads.dataset.sampled_dataset`, 621  
`ads.dataset.target`, 622  
`ads.dataset.timeseries`, 623  
`ads.evaluations`, 633  
`ads.evaluations.evaluation_plot`, 623  
`ads.evaluations.evaluator`, 625  
`ads.evaluations.statistical_metrics`, 631  
`ads.feature_engineering`, 714  
`ads.feature_engineering.accessor.dataframe_accessor`, 638  
`ads.feature_engineering.accessor.mixin.correlation`, 646  
`ads.feature_engineering.accessor.mixin.eda_mixin`, 646  
`ads.feature_engineering.accessor.mixin.eda_mixin_series`, 649  
`ads.feature_engineering.accessor.mixin.feature_types_mixin`, 650  
`ads.feature_engineering.accessor.series_accessor`, 643  
`ads.feature_engineering.adsstring.common_regex_mixin`, 652  
`ads.feature_engineering.adsstring.oci_language`, 653  
`ads.feature_engineering.adsstring.string`, 653  
`ads.feature_engineering.exceptions`, 633  
`ads.feature_engineering.feature_type.address`, 653  
`ads.feature_engineering.feature_type.base`, 656  
`ads.feature_engineering.feature_type.boolean`, 657  
`ads.feature_engineering.feature_type.category`, 659  
`ads.feature_engineering.feature_type.constant`, 662  
`ads.feature_engineering.feature_type.continuous`, 664  
`ads.feature_engineering.feature_type.creditcard`, 666  
`ads.feature_engineering.feature_type.datetime`, 670  
`ads.feature_engineering.feature_type.discrete`, 673  
`ads.feature_engineering.feature_type.document`, 675  
`ads.feature_engineering.feature_type.gis`, 676  
`ads.feature_engineering.feature_type.handler.feature_v`, 705  
`ads.feature_engineering.feature_type.handler.feature_w`, 710  
`ads.feature_engineering.feature_type.handler.warnings`, 713  
`ads.feature_engineering.feature_type.integer`, 680  
`ads.feature_engineering.feature_type.ip_address`, 682  
`ads.feature_engineering.feature_type.ip_address_v4`, 684  
`ads.feature_engineering.feature_type.ip_address_v6`, 686  
`ads.feature_engineering.feature_type.lat_long`, 689  
`ads.feature_engineering.feature_type.object`, 692  
`ads.feature_engineering.feature_type.ordinal`, 693  
`ads.feature_engineering.feature_type.phone_number`, 695  
`ads.feature_engineering.feature_type.string`, 698  
`ads.feature_engineering.feature_type.text`, 700  
`ads.feature_engineering.feature_type.unknown`, 702  
`ads.feature_engineering.feature_type.zip_code`, 703  
`ads.feature_engineering.feature_type_manager`, 634  
`ads.hpo`, 729  
`ads.hpo.distributions`, 714  
`ads.hpo.search_cv`, 717  
`ads.hpo.stopping_criterion`, 728  
`ads.jobs`, 763  
`ads.jobs.ads_job`, 729  
`ads.jobs.builders.infrastructure.dataflow`, 744  
`ads.jobs.builders.infrastructure.dsc_job`, 753  
`ads.jobs.builders.runtimes.python_runtime`, 735  
`ads.model`, 800  
`ads.model.artifact`, 763  
`ads.model.deployment`, 813  
`ads.model.deployment.model_deployer`, 800  
`ads.model.deployment.model_deployment`, 804  
`ads.model.deployment.model_deployment_properties`, 809

- `ads.model.extractor.automl_extractor`, 791
- `ads.model.extractor.keras_extractor`, 797
- `ads.model.extractor.lightgbm_extractor`, 793
- `ads.model.extractor.model_info_extractor`, 794
- `ads.model.extractor.model_info_extractor_factory`, 790
- `ads.model.extractor.pytorch_extractor`, 799
- `ads.model.extractor.sklearn_extractor`, 795
- `ads.model.extractor.tensorflow_extractor`, 798
- `ads.model.extractor.xgboost_extractor`, 792
- `ads.model.framework`, 843
- `ads.model.framework.automl_model`, 813
- `ads.model.framework.lightgbm_model`, 818
- `ads.model.framework.pytorch_model`, 823
- `ads.model.framework.sklearn_model`, 828
- `ads.model.framework.tensorflow_model`, 833
- `ads.model.framework.xgboost_model`, 838
- `ads.model.generic_model`, 765
- `ads.model.model_properties`, 788
- `ads.model.runtime`, 847
- `ads.model.runtime.env_info`, 843
- `ads.model.runtime.model_deployment_details`, 845
- `ads.model.runtime.model_provenance_details`, 845
- `ads.model.runtime.runtime_info`, 790, 846
- `ads.model.runtime.utils`, 846
- `ads.secrets`, 865
- `ads.secrets.adb`, 851
- `ads.secrets.auth_token`, 863
- `ads.secrets.big_data_service`, 859
- `ads.secrets.mysql_db`, 855
- `ads.secrets.oracle_db`, 857
- `ads.secrets.secrets`, 847
- `ads.text_dataset`, 875
- `ads.text_dataset.backends`, 865
- `ads.text_dataset.dataset`, 867
- `ads.text_dataset.extractor`, 872
- `ads.text_dataset.options`, 874
- `ads.vault`, 876
- `ads.vault.vault`, 875
- MULTI\_CLASS\_CLASSIFICATION  
(`ads.common.utils.ml_task_types` attribute), 536
- MULTI\_CLASS\_TEXT\_CLASSIFICATION  
(`ads.common.utils.ml_task_types` attribute), 536
- MULTI\_LABEL (`ads.data_labeling.constants.AnnotationType` attribute), 544
- MultiClassClassificationDataset (class in `ads.dataset.classification_dataset`), 587
- MultiClassTextClassificationDataset (class in `ads.dataset.classification_dataset`), 587
- MultiLabelRecordParser (class in `ads.data_labeling.parser.export_record_parser`), 554
- MYSQL (`ads.common.decorator.runtime_dependency.OptionalDependency` attribute), 522
- MySQLDBSecret (class in `ads.secrets.mysql_db`), 855
- MySQLDBSecretKeeper (class in `ads.secrets.mysql_db`), 855
- N
- `n_trials` (`ads.hpo.search_cv.ADSTuner` property), 720
- `name` (`ads.dataset.helper.ElaboratedPath` property), 616
- `name` (`ads.feature_engineering.accessor.series_accessor.ADSSeriesAccessor` attribute), 643
- `name` (`ads.feature_engineering.feature_type.address.Address` attribute), 653
- `name` (`ads.feature_engineering.feature_type.base.FeatureType` attribute), 656
- `name` (`ads.feature_engineering.feature_type.boolean.Boolean` attribute), 657
- `name` (`ads.feature_engineering.feature_type.category.Category` attribute), 660
- `name` (`ads.feature_engineering.feature_type.constant.Constant` attribute), 662
- `name` (`ads.feature_engineering.feature_type.continuous.Continuous` attribute), 664
- `name` (`ads.feature_engineering.feature_type.creditcard.CreditCard` attribute), 666
- `name` (`ads.feature_engineering.feature_type.datetime.DateTime` attribute), 670
- `name` (`ads.feature_engineering.feature_type.discrete.Discrete` attribute), 673
- `name` (`ads.feature_engineering.feature_type.document.Document` attribute), 675
- `name` (`ads.feature_engineering.feature_type.gis.GIS` attribute), 676
- `name` (`ads.feature_engineering.feature_type.integer.Integer` attribute), 680
- `name` (`ads.feature_engineering.feature_type.ip_address.IpAddress` attribute), 682
- `name` (`ads.feature_engineering.feature_type.ip_address_v4.IpAddressV4` attribute), 684
- `name` (`ads.feature_engineering.feature_type.ip_address_v6.IpAddressV6` attribute), 686
- `name` (`ads.feature_engineering.feature_type.lat_long.LatLong` attribute), 689
- `name` (`ads.feature_engineering.feature_type.object.Object` attribute), 693

**name** (*ads.feature\_engineering.feature\_type.ordinal.Ordinal* attribute), 693  
**name** (*ads.feature\_engineering.feature\_type.phone\_number.PhoneNumber* attribute), 696  
**name** (*ads.feature\_engineering.feature\_type.string.String* attribute), 698  
**name** (*ads.feature\_engineering.feature\_type.text.Text* attribute), 700  
**name** (*ads.feature\_engineering.feature\_type.unknown.Unknown* attribute), 702  
**name** (*ads.feature\_engineering.feature\_type.zip\_code.ZipCode* attribute), 703  
**name** (*ads.jobs.ads\_job.Job* property), 733  
**name** (*ads.jobs.builders.infrastructure.dataflow.DataFlow* property), 745  
**name** (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob* property), 758  
**Name** (class in *ads.feature\_engineering.feature\_type.base*), 656  
**NameAlreadyRegistered**, 633  
**NEEDSACTION** (*ads.model.generic\_model.ModelState* attribute), 787  
**NERItem** (class in *ads.data\_labeling.ner*), 548  
**NERItems** (class in *ads.data\_labeling.ner*), 549  
**NERRecordParser** (class in *ads.data\_labeling.parser.export\_record\_parser*), 554  
**NoRestartError**, 728  
**normalize\_hyperparameter()** (in module *ads.model.extractor.model\_info\_extractor*), 795  
**NOT\_PASSED** (*ads.common.model\_introspect.TEST\_STATUS* attribute), 526  
**NOT\_TESTED** (*ads.common.model\_introspect.TEST\_STATUS* attribute), 526  
**NotActiveDeploymentError**, 788  
**NOTAVAILABLE** (*ads.model.generic\_model.ModelState* attribute), 787  
**NOTEBOOK** (*ads.common.decorator.runtime\_dependency.OptionalDependency* attribute), 522  
**notebook\_encoding** (*ads.jobs.builders.runtimes.python\_runtime.NotebookRuntime* property), 741  
**notebook\_uri** (*ads.jobs.builders.runtimes.python\_runtime.NotebookRuntime* property), 741  
**NotebookCatalog** (class in *ads.catalog.notebook*), 501  
**NotebookRuntime** (class in *ads.jobs.builders.runtimes.python\_runtime*), 740  
**NotebookSummaryList** (class in *ads.catalog.notebook*), 503  
**NTrials** (class in *ads.hpo.stopping\_criterion*), 728  
**num\_paths** (*ads.dataset.helper.ElaboratedPath* property), 616  
**numeric\_pandas\_dtypes()** (in module *ads.common.utils*), 536  
**Object** (class in *ads.feature\_engineering.feature\_type.object*), 692  
**oci\_cli\_auth** (*ads.common.auth.AuthState* attribute), 509  
**oci\_config** (*ads.common.auth.AuthState* attribute), 509  
**oci\_config\_file()** (in module *ads.common.utils*), 536  
**oci\_config\_location()** (in module *ads.common.utils*), 536  
**oci\_config\_path** (*ads.common.auth.AuthState* attribute), 509  
**oci\_config\_profile()** (in module *ads.common.utils*), 536  
**oci\_iam\_type** (*ads.common.auth.AuthState* attribute), 509  
**oci\_key\_location()** (in module *ads.common.utils*), 536  
**oci\_key\_profile** (*ads.common.auth.AuthState* attribute), 509  
**oci\_key\_profile()** (in module *ads.common.utils*), 536  
**oci\_link** (*ads.dataflow.dataflow.DataFlowApp* property), 579  
**oci\_link** (*ads.dataflow.dataflow.DataFlowRun* property), 582  
**oci\_link** (*ads.dataflow.dataflow.RunObserver* property), 583  
**oci\_path** (*ads.dataflow.dataflow.DataFlowLog* property), 581  
**oci\_signer** (*ads.common.auth.AuthState* attribute), 509  
**oci\_signer\_callable** (*ads.common.auth.AuthState* attribute), 509  
**oci\_signer\_kwargs** (*ads.common.auth.AuthState* attribute), 509  
**OCIAuthContext** (class in *ads.common.auth*), 510  
**offset** (*ads.data\_labeling.ner.NERItem* attribute), 548  
**ONNX** (*ads.common.decorator.runtime\_dependency.OptionalDependency* attribute), 522  
**OPCTL** (*ads.common.decorator.runtime\_dependency.OptionalDependency* attribute), 522  
**open()** (*ads.dataset.dataset\_browser.DatasetBrowser* method), 603  
**open()** (*ads.dataset.dataset\_browser.GitHubDatasets* method), 603  
**open()** (*ads.dataset.dataset\_browser.LocalFilesystemDatasets* method), 604  
**open()** (*ads.dataset.dataset\_browser.SeabornDatasets* method), 604  
**open()** (*ads.dataset.dataset\_browser.SklearnDatasets* method), 604  
**open()** (*ads.dataset.dataset\_browser.WebDatasets* method), 605

`open()` (`ads.dataset.factory.DatasetFactory` static method), 612  
`open()` (in module `ads.config`), 876  
`open_to_pandas()` (`ads.dataset.factory.DatasetFactory` static method), 614  
`optimizer()` (`ads.hpo.search_cv.ADSTuner` static method), 720  
`option()` (`ads.text_dataset.dataset.DataLoader` method), 870  
`option_handler()` (`ads.text_dataset.options.OptionFactory` static method), 874  
`option_handlers` (`ads.text_dataset.options.OptionFactory` attribute), 874  
`OptionalDependency` (class in `ads.common.decorator.runtime_dependency`), 522  
`OptionFactory` (class in `ads.text_dataset.options`), 874  
`OptionHandler` (class in `ads.text_dataset.options`), 874  
`Options` (class in `ads.text_dataset.options`), 874  
`OPTUNA` (`ads.common.decorator.runtime_dependency.OptionFactory` attribute), 522  
`OracleAutoMLProvider` (class in `ads.automl.provider`), 492  
`OracleConnector` (class in `ads.database.connection`), 575  
`OracleDBSecret` (class in `ads.secrets.oracledb`), 857  
`OracleDBSecretKeeper` (class in `ads.secrets.oracledb`), 857  
`Ordinal` (class in `ads.feature_engineering.feature_type.ordinal`), 693  
`output_uri` (`ads.jobs.builders.runtimes.python_runtime.NeptuneRuntime` property), 741  
`overwrite_existing_artifact` (`ads.model.model_properties.ModelProperties` attribute), 789

## P

`PACK_TYPE` (class in `ads.model.runtime.env_info`), 844  
`PandasDataset` (class in `ads.dataset.sampled_dataset`), 621  
`parse()` (`ads.data_labeling.interface.parser.Parser` method), 541  
`parse()` (`ads.data_labeling.parser.export_metadata_parser.ExportMetadataParser` static method), 553  
`parse()` (`ads.data_labeling.parser.export_record_parser.RecordParser` method), 555  
`parse_apache_log_datetime()` (in module `ads.dataset.helper`), 617  
`parse_apache_log_str()` (in module `ads.dataset.helper`), 617  
`Parser` (class in `ads.data_labeling.interface.parser`), 541  
`parser()` (`ads.data_labeling.parser.export_record_parser.RecordParserFactory` static method), 556  
`PASSED` (`ads.common.model_introspect.TEST_STATUS` attribute), 526  
`password` (`ads.secrets.adb.ADBSecret` attribute), 851  
`password` (`ads.secrets.mysqladb.MySQLDBSecret` attribute), 855  
`password` (`ads.secrets.oracledb.OracleDBSecret` attribute), 857  
`path` (`ads.data_labeling.record.Record` attribute), 549, 550  
`paths` (`ads.dataset.helper.ElaboratedPath` property), 616  
`payload_attribute_map` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` attribute), 758  
`PDFPlumber` (class in `ads.text_dataset.backends`), 866  
`PDFProcessor` (class in `ads.text_dataset.extractor`), 873  
`pearson()` (`ads.feature_engineering.accessor.mixin.eda_mixin.EDAMixin` method), 648  
`pearson_plot()` (`ads.feature_engineering.accessor.mixin.eda_mixin.EDA` method), 649  
`perfect_kwargs` (`ads.evaluations.evaluation_plot.EvaluationPlot` attribute), 623, 624  
`perfect_kwargs` (`ads.evaluations.evaluation_plot.EvaluationPlot` attribute), 623, 624  
`phone_number_US` (`ads.feature_engineering.adsstring.common_regex_mixin` property), 652  
`PhoneNumber` (class in `ads.feature_engineering.feature_type.phone_number`), 695  
`plot()` (`ads.dataset.sampled_dataset.PandasDataset` method), 621  
`plot()` (`ads.dataset.timeseries.Timeseries` method), 623  
`plot()` (`ads.evaluations.evaluation_plot.EvaluationPlot` class method), 624  
`plot()` (`ads.evaluations.evaluation_plot.EvaluationPlot` method), 623  
`plot_best_scores()` (`ads.hpo.search_cv.ADSTuner` method), 720  
`plot_contour_scores()` (`ads.hpo.search_cv.ADSTuner` method), 721  
`plot_correlation_heatmap()` (`ads.dataset.correlation_plot.BokehHeatMap` method), 589  
`plot_correlation_heatmap()` (in module `ads.dataset.correlation_plot`), 590  
`plot_pdf_scores()` (`ads.hpo.search_cv.ADSTuner` method), 721  
`plot_gis_scatter()` (`ads.dataset.sampled_dataset.PandasDataset` method), 622  
`plot_hbar()` (`ads.dataset.correlation_plot.BokehHeatMap` method), 589  
`plot_heat_map()` (`ads.dataset.correlation_plot.BokehHeatMap` method), 589  
`plot_intermediate_scores()` (`ads.hpo.search_cv.ADSTuner` method), 721



`plot_parallel_coordinate_scores()` (*ads.hpo.search\_cv.ADSTuner* method), 721  
`plot_param_importance()` (*ads.hpo.search\_cv.ADSTuner* method), 722  
`Plotting` (class in *ads.dataset.plot*), 619  
`populate_metadata()` (*ads.common.model\_metadata\_mixin.MetadataMixin* method), 538  
`populate_schema()` (*ads.common.model\_metadata\_mixin.MetadataMixin* method), 539  
`port` (*ads.secrets.mysql\_db.MySQLDBSecret* attribute), 855  
`port` (*ads.secrets.oracle\_db.OracleDBSecret* attribute), 857  
`positive_class` (*ads.evaluations.evaluator.ADSEvaluator* attribute), 625  
`positive_class` (*ads.evaluations.statistical\_metrics.ModelMetrics* attribute), 631  
`Positive_Class_Names` (*ads.evaluations.evaluator.ADSEvaluator* attribute), 627  
`Positive_Class_names` (*ads.evaluations.evaluator.ADSEvaluator* attribute), 626  
`precision` (*ads.evaluations.evaluator.ADSEvaluator.EvaluationMetric* property), 627  
`PREDICT` (*ads.model.deployment.model\_deployment.ModelDeployment* attribute), 809  
`predict()` (*ads.automl.provider.BaselineModel* method), 491  
`predict()` (*ads.common.model.ADSModel* method), 519  
`predict()` (*ads.model.deployment.model\_deployment.ModelDeployment* method), 807  
`predict()` (*ads.model.framework.automl\_model.AutoMLModel* method), 816  
`predict()` (*ads.model.framework.lightgbm\_model.LightGBMModel* method), 820  
`predict()` (*ads.model.framework.pytorch\_model.PyTorchModel* method), 825  
`predict()` (*ads.model.framework.sklearn\_model.SklearnModel* method), 830  
`predict()` (*ads.model.framework.tensorflow\_model.TensorFlowModel* method), 835  
`predict()` (*ads.model.framework.xgboost\_model.XGBoostModel* method), 840  
`predict()` (*ads.model.generic\_model.FrameworkSpecificModel* method), 766  
`predict()` (*ads.model.generic\_model.GenericModel* method), 770, 777  
`predict_log` (*ads.model.deployment.model\_deployment.ModelDeployment* property), 808  
`predict_proba()` (*ads.automl.provider.BaselineModel* method), 491  
`predict_proba()` (*ads.common.model.ADSModel* method), 519  
`prepare()` (*ads.common.model.ADSModel* method), 519  
`prepare()` (*ads.model.framework.automl\_model.AutoMLModel* method), 816  
`prepare()` (*ads.model.framework.lightgbm\_model.LightGBMModel* method), 820  
`prepare()` (*ads.model.framework.pytorch\_model.PyTorchModel* method), 825  
`prepare()` (*ads.model.framework.sklearn\_model.SklearnModel* method), 830  
`prepare()` (*ads.model.framework.tensorflow\_model.TensorFlowModel* method), 835  
`prepare()` (*ads.model.framework.xgboost\_model.XGBoostModel* method), 840  
`prepare_app()` (*ads.dataflow.dataflow.DataFlow* method), 577  
`prepare_fn_attributes()` (in module *ads.common.function.fn\_util*), 529  
`prepare_generic_model()` (in module *ads.common.model\_export\_util*), 526  
`prepare_metrics()` (*ads.dataflow.dataflow.DataFlowApp* method), 579  
`prepare_runtime_yaml()` (*ads.model.artifact.ModelArtifact* method), 764  
`prepare_save_deploy()` (*ads.model.generic\_model.GenericModel* method), 770, 780  
`prepare_score_py()` (*ads.model.artifact.ModelArtifact* method), 765  
`prepare_string_type` (*ads.feature\_engineering.adsstring.common\_regex\_mixin.CommonRegexMixin* property), 652  
`Principal` (*ads.secrets.big\_data\_service.BDSecret* attribute), 859, 860  
`Principal` (*ads.secrets.big\_data\_service.BDSecretKeeper* attribute), 860  
`print_summary()` (*ads.automl.provider.OracleAutoMLProvider* method), 492  
`Print_Models()` (*ads.automl.provider.OracleAutoMLProvider* method), 492  
`Print_user_message()` (in module *ads.common.utils*), 536  
`PrintItem` (class in *ads.common.model\_introspect*), 525  
`prob_type` (*ads.evaluations.evaluation\_plot.EvaluationPlot* attribute), 623, 624  
`processor` (*ads.text\_dataset.dataset.DataLoader* attribute), 867  
`processor_map` (*ads.text\_dataset.extractor.FileProcessorFactory* attribute), 873  
`ProgressBar` (class in *ads.dataset.progress*), 620

[project\\_id\(ads.jobs.builders.infrastructure.dsc\\_job.DataSearchJob.data\\_labeling.reader.metadata\\_reader.ExportMetadataReader property\), 758](#)  
[project\\_id\(ads.model.model\\_properties.ModelProperties.read\(\) \(ads.data\\_labeling.reader.metadata\\_reader.MetadataReader attribute\), 789](#)  
[project\\_ocid\(ads.model.runtime.model\\_provenance\\_details.ModelProvenanceDetails.reader.record\\_reader.RecordReader attribute\), 845](#)  
[ProjectCatalog \(class in ads.catalog.project\), 503](#)  
[ProjectSummaryList \(class in ads.catalog.project\), 505](#)  
[properties\(ads.model.deployment.model\\_deployment.ModelDeploymentModel attribute\), 805](#)  
[properties\(ads.model.framework.automl\\_model.AutoMLModel attribute\), 815](#)  
[properties\(ads.model.framework.lightgbm\\_model.LightGBMModel attribute\), 819](#)  
[properties\(ads.model.framework.pytorch\\_model.PyTorchModel attribute\), 824](#)  
[properties\(ads.model.framework.sklearn\\_model.SklearnModel attribute\), 829](#)  
[properties\(ads.model.framework.tensorflow\\_model.TensorFlowModel attribute\), 834](#)  
[properties\(ads.model.framework.xgboost\\_model.XGBoostModel attribute\), 840](#)  
[properties \(ads.model.generic\\_model.GenericModel attribute\), 769](#)  
[PythonRuntime \(class in ads.jobs.builders.runtimes.python\\_runtime\), 742](#)  
[PYTORCH\(ads.common.decorator.runtime\\_dependency.OptionalDependency attribute\), 522](#)  
[PyTorchExtractor \(class in ads.model.extractor.pytorch\\_extractor\), 799](#)  
[PytorchExtractor \(class in ads.model.extractor.pytorch\\_extractor\), 800](#)  
[PyTorchModel \(class in ads.model.framework.pytorch\\_model\), 823](#)  
**R**  
[random\\_valid\\_ocid\(\) \(in module ads.common.utils\), 537](#)  
[raw\\_metrics \(ads.evaluations.evaluator.ADSEvaluator property\), 630](#)  
[read\(\) \(ads.data\\_labeling.interface.reader.Reader method\), 541](#)  
[read\(\) \(ads.data\\_labeling.reader.dataset\\_reader.DLSDatasetReader method\), 558, 559](#)  
[read\(\) \(ads.data\\_labeling.reader.dataset\\_reader.ExportReader method\), 559, 560](#)  
[read\(\) \(ads.data\\_labeling.reader.dataset\\_reader.LabeledDatasetReader method\), 560, 563](#)  
[read\(\) \(ads.data\\_labeling.reader.jsonl\\_reader.JsonlReader method\), 564](#)  
[read\(\) \(ads.data\\_labeling.reader.metadata\\_reader.DLSMetadataReader method\), 564](#)  
[read\(\) \(ads.data\\_labeling.reader.metadata\\_reader.ExportMetadataReader method\), 565](#)  
[read\(\) \(ads.data\\_labeling.reader.metadata\\_reader.MetadataReader method\), 566](#)  
[read\(\) \(ads.data\\_labeling.reader.record\\_reader.RecordReader method\), 569](#)  
[read\\_arff\(\) \(ads.dataset.factory.CustomFormatReaders static method\), 610](#)  
[read\\_avro\(\) \(ads.dataset.factory.CustomFormatReaders static method\), 610](#)  
[read\\_html\(\) \(ads.dataset.factory.CustomFormatReaders static method\), 610](#)  
[read\\_json\(\) \(ads.dataset.factory.CustomFormatReaders static method\), 610](#)  
[read\\_labeled\\_data\(\) \(ads.data\\_labeling.mixin.data\\_labeling.DataLabelingAccessMixin static method\), 550](#)  
[read\\_libsvm\(\) \(ads.dataset.factory.CustomFormatReaders static method\), 611](#)  
[read\\_line\(\) \(ads.text\\_dataset.backends.Base method\), 865](#)  
[read\\_line\(\) \(ads.text\\_dataset.backends.PDFPlumber method\), 866](#)  
[read\\_line\(\) \(ads.text\\_dataset.backends.Tika method\), 867](#)  
[read\\_line\(\) \(ads.text\\_dataset.dataset.DataLoader method\), 870](#)  
[read\\_line\(\) \(ads.text\\_dataset.extractor.FileProcessor method\), 872](#)  
[read\\_log\(\) \(ads.dataset.factory.CustomFormatReaders static method\), 611](#)  
[read\\_sql\(\) \(ads.dataset.factory.CustomFormatReaders class method\), 611](#)  
[read\\_text\(\) \(ads.text\\_dataset.backends.Base method\), 865](#)  
[read\\_text\(\) \(ads.text\\_dataset.backends.PDFPlumber method\), 866](#)  
[read\\_text\(\) \(ads.text\\_dataset.backends.Tika method\), 867](#)  
[read\\_text\(\) \(ads.text\\_dataset.dataset.DataLoader method\), 871](#)  
[read\\_text\(\) \(ads.text\\_dataset.extractor.FileProcessor method\), 873](#)  
[read\\_tsv\(\) \(ads.dataset.factory.CustomFormatReaders static method\), 611](#)  
[read\\_xml\(\) \(ads.dataset.factory.CustomFormatReaders static method\), 611](#)  
[ReadDatasetError, 567](#)  
[Reader \(class in ads.data\\_labeling.interface.reader\), 541](#)  
[Recommendation \(class in ads.dataset.recommendation\), 620](#)  
[recommendation\\_type\\_labels \(ads.dataset.recommendation.Recommendation attribute\), 620](#)

**recommendation\_types** (ads.dataset.recommendation.Recommendation attribute), 620  
**RecommendationTransformer** (class in ads.dataset.recommendation\_transformer), 620  
**Record** (class in ads.data\_labeling.record), 549  
**RecordParser** (class in ads.data\_labeling.parser.export\_record\_parser), 555  
**RecordParserFactory** (class in ads.data\_labeling.parser.export\_record\_parser), 556  
**RecordReader** (class in ads.data\_labeling.reader.record\_reader), 567  
**records\_path** (ads.data\_labeling.metadata.Metadata attribute), 546, 547  
**redact()** (ads.feature\_engineering.adsstring.common\_regex\_mixin.CommonRegexMixin method), 652  
**redact\_map** (ads.feature\_engineering.adsstring.common\_regex\_mixin.CommonRegexMixin attribute), 653  
**refresh\_ticket()** (in module ads.bds.auth), 540  
**register()** (ads.common.auth.AuthFactory class method), 508  
**register()** (ads.data\_labeling.parser.export\_record\_parser.ExportRecordParser class method), 556  
**register()** (ads.feature\_engineering.feature\_type.handler.feature\_type\_handler.FeatureTypeHandler method), 705, 707  
**register()** (ads.feature\_engineering.feature\_type.handler.feature\_type\_handler.FeatureTypeHandler method), 708  
**register()** (ads.feature\_engineering.feature\_type.handler.feature\_type\_handler.FeatureTypeHandler method), 711, 712  
**register()** (ads.text\_dataset.extractor.FileProcessorFactory class method), 873  
**register\_option()** (ads.text\_dataset.options.OptionFactory class method), 874  
**registered()** (ads.feature\_engineering.feature\_type.handler.feature\_type\_handler.FeatureTypeHandler method), 706, 707  
**registered()** (ads.feature\_engineering.feature\_type.handler.feature\_type\_handler.FeatureTypeHandler method), 708, 709  
**registered()** (ads.feature\_engineering.feature\_type.handler.feature\_type\_handler.FeatureTypeHandler method), 711, 712  
**REGRESSION** (ads.common.utils.ml\_task\_types attribute), 536  
**RegressionDataset** (class in ads.dataset.regression\_dataset), 621  
**reload()** (ads.model.artifact.ModelArtifact method), 765  
**reload()** (ads.model.framework.automl\_model.AutoMLModel method), 816  
**reload()** (ads.model.framework.lightgbm\_model.LightGBMModel method), 820  
**reload()** (ads.model.framework.pytorch\_model.PyTorchModel method), 825  
**reload()** (ads.model.framework.sklearn\_model.SklearnModel method), 830  
**reload()** (ads.model.framework.tensorflow\_model.TensorFlowModel method), 835  
**reload()** (ads.model.framework.xgboost\_model.XGBoostModel method), 841  
**reload()** (ads.model.generic\_model.GenericModel method), 770, 783  
**reload\_runtime\_info()** (ads.model.generic\_model.GenericModel method), 783  
**remove\_existing\_artifact** (ads.model.model\_properties.ModelProperties attribute), 789  
**remove\_file()** (in module ads.common.utils), 537  
**rename()** (ads.common.model.ADSModel method), 520  
**rename\_columns()** (ads.dataset.dataset.ADSDataSet method), 594  
**rename\_columns()** (ads.dataset.dataset\_with\_target.ADSDataSetWithTarget method), 624  
**rename\_duplicate\_cols()** (in module ads.dataset.helper), 618  
**render()** (ads.data\_labeling.visualizer.text\_visualizer.TextLabeledDataForTextVisualizer static method), 574  
**render()** (ads.data\_labeling.visualizer.image\_visualizer.ImageLabeledDataForImageVisualizer static method), 569, 571  
**render()** (ads.data\_labeling.visualizer.text\_visualizer.TextLabeledDataForTextVisualizer static method), 572, 574  
**render\_bound\_in\_feature()** (ads.data\_labeling.mixin.data\_labeling.DataLabelingAccessMixin static method), 570  
**render\_item()** (ads.data\_labeling.visualizer.image\_visualizer.ImageLabeledDataForImageVisualizer static method), 570  
**render\_ner()** (ads.data\_labeling.mixin.data\_labeling.DataLabelingAccessMixin static method), 552  
**RenderOptions** (class in ads.data\_labeling.visualizer.image\_visualizer), 571  
**RenderOptions** (class in ads.data\_labeling.visualizer.text\_visualizer), 572  
**render\_warning()** (ads.data\_labeling.mixin.data\_labeling.DataLabelingAccessMixin static method), 570  
**replace\_spaces()** (in module ads.common.utils), 537  
**required\_keys** (ads.secrets.secrets.SecretKeeper attribute), 850  
**RESOURCE\_PRINCIPAL** (ads.common.auth.AuthType attribute), 509  
**resource\_principal()** (in module ads.common.auth), 514  
**ResourcePrincipal** (class in ads.common.auth), 511  
**response** (ads.catalog.project.ProjectSummaryList attribute), 505  
**result** (ads.common.model\_introspect.PrintItem attribute), 526



resume() (*ads.hpo.search\_cv.ADSTuner* method), 722  
 rollback() (*ads.catalog.model.Model* method), 494, 496  
 run() (*ads.common.model\_introspect.ModelIntrospect* method), 524, 525  
 run() (*ads.dataflow.dataflow.DataFlowApp* method), 580  
 run() (*ads.jobs.ads\_job.Job* method), 733  
 run() (*ads.jobs.builders.infrastructure.dataflow.DataFlow* method), 746  
 run() (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob* method), 758  
 run() (*ads.jobs.builders.infrastructure.dsc\_job.DSCJob* method), 754  
 run\_details\_link(*ads.jobs.builders.infrastructure.dataflow.DataFlow* property), 751  
 run\_list() (*ads.jobs.ads\_job.Job* method), 733  
 run\_list() (*ads.jobs.builders.infrastructure.dataflow.DataFlow* method), 746  
 run\_list() (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob* method), 758  
 run\_list() (*ads.jobs.builders.infrastructure.dsc\_job.DSCJob* method), 755  
 RUNNING (*ads.hpo.search\_cv.State* attribute), 728  
 RunObserver (class in *ads.dataflow.dataflow*), 583  
 runtime (*ads.jobs.ads\_job.Job* property), 734  
 runtime\_dependency() (in module *ads.common.decorator.runtime\_dependency*), 522  
 runtime\_info(*ads.model.framework.automl\_model.AutoMLModel* attribute), 819  
 runtime\_info(*ads.model.framework.lightgbm\_model.LightGBMModel* attribute), 819  
 runtime\_info(*ads.model.framework.pytorch\_model.PyTorchModel* attribute), 824  
 runtime\_info(*ads.model.framework.sklearn\_model.SklearnModel* attribute), 829  
 runtime\_info(*ads.model.framework.tensorflow\_model.TensorFlowModel* attribute), 835  
 runtime\_info(*ads.model.framework.xgboost\_model.XGBoostModel* attribute), 840  
 runtime\_info(*ads.model.generic\_model.GenericModel* attribute), 769  
 RuntimeInfo (class in *ads.model.runtime.runtime\_info*), 790, 846  
 RuntimeInfoInconsistencyError, 788

## S

safe\_metrics\_call()  
     (*ads.evaluations.statistical\_metrics.ModelEvaluator* method), 631, 632  
 sample() (*ads.dataset.dataset.ADSDataset* method), 594  
 sampling\_confidence\_interval  
     (*ads.dataset.helper.DatasetDefaults* attribute), 616  
 sampling\_confidence\_level  
     (*ads.dataset.helper.DatasetDefaults* attribute), 616  
 save() (*ads.dataflow.dataflow.DataFlowLog* method), 581  
 save() (*ads.model.framework.automl\_model.AutoMLModel* method), 816  
 save() (*ads.model.framework.lightgbm\_model.LightGBMModel* method), 820  
 save() (*ads.model.framework.pytorch\_model.PyTorchModel* method), 825  
 save() (*ads.model.framework.sklearn\_model.SklearnModel* method), 830  
 save() (*ads.model.framework.tensorflow\_model.TensorFlowModel* method), 835  
 save() (*ads.model.framework.xgboost\_model.XGBoostModel* method), 841  
 save() (*ads.model.generic\_model.GenericModel* method), 770, 783  
 save() (*ads.model.runtime.runtime\_info.RuntimeInfo* method), 790, 846  
 save() (*ads.secrets.adb.ADBSecretKeeper* method), 854  
 save() (*ads.secrets.big\_data\_service.BDSSecretKeeper* method), 863  
 save() (*ads.secrets.secrets.SecretKeeper* method), 850  
 schema\_input(*ads.model.framework.automl\_model.AutoMLModel* attribute), 815  
 schema\_input(*ads.model.framework.lightgbm\_model.LightGBMModel* attribute), 824  
 schema\_input(*ads.model.framework.pytorch\_model.PyTorchModel* attribute), 830  
 schema\_input(*ads.model.framework.sklearn\_model.SklearnModel* attribute), 835  
 schema\_input(*ads.model.framework.tensorflow\_model.TensorFlowModel* attribute), 840  
 schema\_input(*ads.model.generic\_model.GenericModel* attribute), 769  
 schema\_input(*ads.model.generic\_model.GenericModel* property), 784  
 schema\_output(*ads.model.framework.automl\_model.AutoMLModel* attribute), 815  
 schema\_output(*ads.model.framework.lightgbm\_model.LightGBMModel* attribute), 819  
 schema\_output(*ads.model.framework.pytorch\_model.PyTorchModel* attribute), 825  
 schema\_output(*ads.model.framework.sklearn\_model.SklearnModel* attribute), 830  
 schema\_output(*ads.model.framework.tensorflow\_model.TensorFlowModel* attribute), 835  
 schema\_output(*ads.model.framework.xgboost\_model.XGBoostModel* attribute), 840

- `schema_output` (*ads.model.generic\_model.GenericModel* attribute), 769
- `schema_output` (*ads.model.generic\_model.GenericModel* property), 784
- `SchemaValidator` (class in *ads.model.runtime.utils*), 846
- `score()` (*ads.common.model.ADSModel* method), 520
- `score_remaining` (*ads.hpo.search\_cv.ADSTuner* property), 722
- `ScoreValue` (class in *ads.hpo.stopping\_criterion*), 728
- `scoring_name` (*ads.hpo.search\_cv.ADSTuner* property), 723
- `script_bucket` (*ads.jobs.builders.runtimes.python\_runtime.DataFlowRuntime* property), 737
- `script_uri` (*ads.jobs.builders.runtimes.python\_runtime.DataFlowRuntime* property), 737
- `script_uri` (*ads.jobs.builders.runtimes.python\_runtime.ScriptRuntime* property), 743
- `ScriptRuntime` (class in *ads.jobs.builders.runtimes.python\_runtime*), 742
- `seaborn()` (*ads.dataset.dataset\_browser.DatasetBrowser* static method), 603
- `SeabornDatasets` (class in *ads.dataset.dataset\_browser*), 604
- `search_space()` (*ads.hpo.search\_cv.ADSTuner* method), 723
- `Secret` (class in *ads.secrets.secrets*), 847
- `secret_id` (*ads.secrets.big\_data\_service.BDSecret* attribute), 860
- `secret_id` (*ads.secrets.big\_data\_service.BDSecretKeeper* attribute), 861
- `SecretKeeper` (class in *ads.secrets.secrets*), 848
- `select_best_features()` (*ads.dataset.classification\_dataset.BinaryTextClassificationDataset* method), 585
- `select_best_features()` (*ads.dataset.classification\_dataset.MultiClassTextClassificationDataset* method), 588
- `select_best_features()` (*ads.dataset.dataset\_with\_target.ADSDatasetWithTarget* method), 607
- `select_best_features()` (*ads.dataset.forecasting\_dataset.ForecastingDataset* method), 616
- `select_best_plot()` (*ads.dataset.plot.Plotting* method), 619
- `selected_model_name()` (*ads.automl.provider.OracleAutoMLProvider* method), 493
- `selected_score_label()` (*ads.automl.provider.OracleAutoMLProvider* method), 493
- `serialize` (*ads.model.framework.automl\_model.AutoMLModel* attribute), 815
- `serialize` (*ads.model.framework.lightgbm\_model.LightGBMModel* attribute), 819
- `serialize` (*ads.model.framework.pytorch\_model.PyTorchModel* attribute), 825
- `serialize` (*ads.model.framework.sklearn\_model.SklearnModel* attribute), 830
- `serialize` (*ads.model.framework.tensorflow\_model.TensorFlowModel* attribute), 835
- `serialize` (*ads.model.framework.xgboost\_model.XGBoostModel* attribute), 840
- `serialize` (*ads.model.generic\_model.GenericModel* attribute), 847
- `serialize()` (*ads.secrets.secrets.Secret* method), 847
- `serialize_model()` (*ads.model.framework.automl\_model.AutoMLModel* method), 822
- `serialize_model()` (*ads.model.framework.lightgbm\_model.LightGBMModel* method), 826
- `serialize_model()` (*ads.model.framework.pytorch\_model.PyTorchModel* method), 832
- `serialize_model()` (*ads.model.framework.sklearn\_model.SklearnModel* method), 832
- `serialize_model()` (*ads.model.framework.tensorflow\_model.TensorFlowModel* method), 837
- `serialize_model()` (*ads.model.framework.xgboost\_model.XGBoostModel* method), 842
- `serialize_model()` (*ads.model.generic\_model.GenericModel* method), 784
- `serialize_model()` (in module *ads.common.model\_export\_util*), 528
- `SerializeInputNotImplementedError`, 788
- `SerializeModelNotImplementedError`, 788
- `service_name` (*ads.secrets.adb.ADBSecret* attribute), 857
- `service_name` (*ads.secrets.oracledb.OracleDBSecret* attribute), 857
- `SERVICE_PACK_DATA` (*ads.model.runtime.env\_info.PACK\_TYPE* attribute), 844
- `set_auth()` (in module *ads.common.auth*), 515
- `set_debug_mode()` (in module *ads*), 877
- `set_default_storage()` (*ads.dataset.factory.DatasetFactory* static method), 614
- `set_description()` (*ads.dataset.dataset.ADSDataset* method), 594
- `set_documentation_mode()` (in module *ads*), 877
- `set_expert_mode()` (in module *ads*), 877
- `set_name()` (*ads.dataset.dataset.ADSDataset* method), 595
- `set_oci_config()` (in module *ads.common.utils*), 537
- `set_positive_class()` (*ads.dataset.classification\_dataset.BinaryClassificationDataset* method), 585

`set_target()` (`ads.dataset.dataset.ADSDataset` method), 556  
`SingletonMeta` (class in `ads.common.auth`), 511  
`setup()` (`ads.automl.provider.AutoMLProvider` method), 490  
`skew_handler()` (in module `ads.feature_engineering.feature_type.handler.warnings`), 713  
`shape_config_details` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` metadata\_update property), 759  
`shape_config_details_attribute_map` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` property), 739  
`shape_name` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` attribute), 759  
`shape_name` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` static method), 603  
`short_id_index` (`ads.catalog.project.ProjectSummaryLists` attribute), 505  
`show_all()` (`ads.dataflow.dataflow.DataFlowLog` method), 581  
`show_corr()` (`ads.dataset.dataset.ADSDataset` method), 595  
`show_deployments()` (`ads.model.deployment.model_deployment.ModelDeployment` method), 801, 803  
`show_full_name` (`ads.evaluations.evaluator.ADSEvaluator` attribute), 625  
`show_in_notebook()` (`ads.catalog.model.Model` method), 494, 496  
`show_in_notebook()` (`ads.catalog.summary.SummaryList` method), 506  
`show_in_notebook()` (`ads.common.model.ADSSModel` method), 521  
`show_in_notebook()` (`ads.dataset.dataset.ADSDataset` method), 596  
`show_in_notebook()` (`ads.dataset.feature_selection.FeatureImportance` method), 615  
`show_in_notebook()` (`ads.dataset.plot.Plotting` method), 619  
`show_in_notebook()` (`ads.dataset.recommendation.Recommendation` method), 620  
`show_in_notebook()` (`ads.dataset.target.TargetVariable` method), 622  
`show_in_notebook()` (`ads.evaluations.evaluator.ADSEvaluator` method), 626, 630  
`show_in_notebook()` (`ads.evaluations.evaluator.ADSEvaluator` method), 627  
`show_logs()` (`ads.model.deployment.model_deployment.ModelDeployment` method), 808  
`sid` (`ads.secrets.oracledb.OracleDBSecret` attribute), 857  
`signerGenerator()` (`ads.common.auth.AuthFactory` method), 509  
`SINGLE_LABEL` (`ads.data_labeling.constants.AnnotationTypes` attribute), 544  
`single_overlay_plots` (`ads.evaluations.evaluation_plot.EvaluationPlot` attribute), 624  
`SingleLabelRecordParser` (class in `ads.data_labeling.parser.export_record_parser`), 805  
`SklearnMeta` (class in `ads.common.auth`), 511  
`sklearn_handler()` (in module `ads.feature_engineering.feature_type.handler.warnings`), 713  
`sklearn_metadata_update` (`ads.jobs.builders.runtimes.python_runtime.GitPythonRuntime` property), 739  
`sklearn()` (`ads.dataset.dataset_browser.DatasetBrowser` static method), 603  
`SklearnDatasets` (`ads.dataset.dataset_browser.SklearnDatasets` attribute), 605  
`sklearn_steps` (`ads.hpo.search_cv.ADSTuner` property), 723  
`SklearnDatasets` (class in `ads.dataset.dataset_browser`), 604  
`SklearnExtractor` (class in `ads.model.extractor.sklearn_extractor`), 795  
`SklearnModelDeployer` (class in `ads.model.framework.sklearn_model`), 828  
`snake_to_camel()` (in module `ads.common.utils`), 537  
`snake_to_camel_map` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` attribute), 759  
`snapshot()` (`ads.dataset.dataset.ADSDataset` method), 597  
`sort_by()` (`ads.catalog.model.ModelSummaryList` method), 500, 501  
`sort_by()` (`ads.catalog.notebook.NotebookSummaryList` method), 503  
`sort_by()` (`ads.catalog.project.ProjectSummaryList` method), 506  
`sort_by()` (`ads.catalog.summary.SummaryList` method), 506  
`sort_by()` (`ads.dataflow.dataflowssummary.SummaryList` method), 584  
`source_path` (`ads.data_labeling.metadata.Metadata` attribute), 546, 548  
`source_uri` (`ads.jobs.builders.runtimes.python_runtime.ScriptRuntime` property), 743  
`SPACY` (`ads.data_labeling.constants.Formats` attribute), 546  
`SPARK` (`ads.common.decorator.runtime_dependency.OptionalDependency` attribute), 522  
`SPARK_VERSION` (class in `ads.dataflow.dataflow`), 584  
`split_data()` (in module `ads.common.utils`), 538  
`ssh_secret_ocid` (`ads.jobs.builders.runtimes.python_runtime.GitPythonRuntime` property), 740  
`ssn` (`ads.feature_engineering.adsstring.common_regex_mixin.CommonRegex` property), 653  
`standardize_spec()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` static method), 759  
`state` (`ads.model.deployment.model_deployment.ModelDeployment` attribute), 805  
`state` (`ads.model.deployment.model_deployment.ModelDeployment` attribute), 805

property), 808

State (class in `ads.hpo.search_cv`), 728

status (ads.common.model\_introspect.ModelIntrospect property), 525

status (ads.dataflow.dataflow.DataFlowRun property), 583

status (ads.dataflow.dataflow.RunObserver property), 583

status (ads.hpo.search\_cv.ADSTuner property), 723

status (ads.jobs.builders.infrastructure.dataflow.DataFlowRun property), 752

status (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob property), 759

status (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun property), 763

status (ads.model.deployment.model\_deployment.ModelDeployment property), 808

status() (ads.jobs.ads\_job.Job method), 734

steps (ads.dataset.pipeline.TransformerPipeline attribute), 619

String (class in ads.feature\_engineering.feature\_type.strings), 698

sub\_properties (ads.model.deployment.model\_deployment\_properties attribute), 811

subnet\_id (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob property), 759

suggest\_recommendations() (ads.dataset.dataset\_with\_target.ADSDatasetWithTarget method), 608

summary() (ads.common.model.ADSModel method), 521

summary() (ads.dataset.sampled\_dataset.PandasDataset method), 622

summary\_status() (ads.model.framework.automl\_model.AutoMLModel method), 816

summary\_status() (ads.model.framework.lightgbm\_model.LightGBMModel method), 820

summary\_status() (ads.model.framework.pytorch\_model.PyTorchModel method), 825

summary\_status() (ads.model.framework.sklearn\_model.SklearnModel method), 830

summary\_status() (ads.model.framework.tensorflow\_model.TensorFlowModel method), 836

summary\_status() (ads.model.framework.xgboost\_model.XGBoostModel method), 841

summary\_status() (ads.model.generic\_model.GenericModel method), 770, 785

SummaryList (class in ads.catalog.summary), 506

SummaryList (class in ads.dataflow.dataflowssummary), 584

SummaryStatus (class in ads.model.generic\_model), 788

swagger\_types (ads.model.deployment.model\_deployment\_properties attribute), 809

sync() (ads.feature\_engineering.accessor.dataframe\_accessor.ADSDatasetAccessor method), 639, 642

sync() (ads.feature\_engineering.accessor.series\_accessor.ADSSeriesAccessor method), 644, 645

## T

Tag (class in ads.feature\_engineering.feature\_type.base), 656

tags (ads.feature\_engineering.accessor.dataframe\_accessor.ADSDatasetAccessor attribute), 639

tags (ads.feature\_engineering.accessor.dataframe\_accessor.ADSDatasetAccessor property), 642

tags (ads.feature\_engineering.accessor.series\_accessor.ADSSeriesAccessor attribute), 643

tail() (ads.dataflow.dataflow.DataFlowLog method), 581

TERMINATE (ads.common.decorator.runtime\_dependency.OptionalDependency attribute), 522

TensorFlowExtractor (class in ads.model.extractor.tensorflow\_extractor), 798

TensorFlowModel (class in ads.model.framework.tensorflow\_model), 833

TERMINAL\_STATES (ads.jobs.builders.infrastructure.dsc\_job.DataScienceJob attribute), 762

terminate() (ads.hpo.search\_cv.ADSTuner method), 728

TERMINATED (ads.hpo.search\_cv.State attribute), 728

TERMINATED\_STATES (ads.jobs.builders.infrastructure.dataflow.DataFlowRun attribute), 751

test\_data (ads.evaluations.evaluator.ADSEvaluator attribute), 625

TEST\_STATES (class in ads.common.model\_introspect), 526

TEXT (ads.common.decorator.runtime\_dependency.OptionalDependency attribute), 522

TEXT (ads.data\_labeling.constants.DatasetType attribute), 544

Text (class in ads.feature\_engineering.feature\_type.text), 700

TextDatasetFactory (class in ads.text\_dataset.dataset), 871

TextLabeledDataFormatter (class in ads.data\_labeling.visualizer.text\_visualizer), 874

TEXTSELECTION (ads.data\_labeling.parser.export\_record\_parser.EntityType attribute), 534



- Tika (class in *ads.text\_dataset.backends*), 866
- time (*ads.feature\_engineering.adsstring.common\_regex\_mixin.CommonRegexMixin* property), 653
- time\_elapsed (*ads.hpo.search\_cv.ADSTuner* property), 724
- time\_remaining (*ads.hpo.search\_cv.ADSTuner* property), 724
- time\_since\_resume (*ads.hpo.search\_cv.ADSTuner* property), 724
- TimeBudget (class in *ads.hpo.stopping\_criterion*), 729
- Timeseries (class in *ads.dataset.timeseries*), 623
- timeseries() (*ads.dataset.sampled\_dataset.PandasDataset* method), 622
- to\_avro() (*ads.dataset.dataset.ADSDataset* method), 597
- to\_csv() (*ads.dataset.dataset.ADSDataset* method), 598
- to\_dask() (*ads.dataset.dataset.ADSDataset* method), 598
- to\_dask\_dataframe() (*ads.dataset.dataset.ADSDataset* method), 599
- to\_dataframe() (*ads.catalog.model.Model* method), 494, 496
- to\_dataframe() (*ads.catalog.summary.SummaryList* method), 506
- to\_dataframe() (*ads.common.model\_introspect.ModelIntrospect* method), 524, 525
- to\_dataframe() (*ads.data\_labeling.metadata.Metadata* method), 548
- to\_dataframe() (*ads.dataflow.dataflowsummary.SummaryList* method), 584
- to\_dataframe() (in module *ads.common.utils*), 538
- to\_dict() (*ads.data\_labeling.metadata.Metadata* method), 548
- to\_dict() (*ads.data\_labeling.record.Record* method), 550
- to\_dict() (*ads.data\_labeling.visualizer.image\_visualizer.ImageVisualizer* method), 571
- to\_dict() (*ads.data\_labeling.visualizer.text\_visualizer.TextVisualizer* method), 573
- to\_dict() (*ads.jobs.ads\_job.Job* method), 734
- to\_dict() (*ads.jobs.builders.infrastructure.dataflow.DataFlow* method), 746
- to\_dict() (*ads.secrets.secrets.Secret* method), 847, 848
- to\_dict() (*ads.secrets.secrets.SecretKeeper* method), 851
- to\_h2o() (*ads.dataset.dataset.ADSDataset* method), 599
- to\_h2o\_dataframe() (*ads.dataset.dataset.ADSDataset* method), 600
- to\_hdf() (*ads.dataset.dataset.ADSDataset* method), 600
- to\_json() (*ads.dataset.dataset.ADSDataset* method), 600
- to\_list() (*ads.common.model\_introspect.PrintItem* method), 526
- to\_oci\_model() (*ads.model.deployment.model\_deployment\_properties.ModelDeploymentProperties* method), 822
- to\_onnx() (*ads.model.framework.lightgbm\_model.LightGBMModel* method), 822
- to\_onnx() (*ads.model.framework.pytorch\_model.PyTorchModel* method), 827
- to\_onnx() (*ads.model.framework.sklearn\_model.SklearnModel* method), 833
- to\_onnx() (*ads.model.framework.tensorflow\_model.TensorFlowModel* method), 837
- to\_onnx() (*ads.model.framework.xgboost\_model.XGBoostModel* method), 843
- to\_onnxrt() (*ads.common.data.ADSData* method), 517
- to\_pandas() (*ads.dataset.dataset.ADSDataset* method), 601
- to\_pandas\_dataframe() (*ads.dataset.dataset.ADSDataset* method), 601
- to\_parquet() (*ads.dataset.dataset.ADSDataset* method), 601
- to\_spacy() (*ads.data\_labeling.ner.NERItem* method), 548
- to\_spacy() (*ads.data\_labeling.ner.NERItems* method), 549
- to\_tuple() (*ads.data\_labeling.record.Record* method), 550
- to\_update\_deployment() (*ads.model.deployment.model\_deployment\_properties.ModelDeploymentProperties* method), 811
- to\_xgb() (*ads.dataset.dataset.ADSDataset* method), 602
- to\_xgb\_dmatrix() (*ads.dataset.dataset.ADSDataset* method), 602
- to\_yaml() (*ads.jobs.builders.infrastructure.dataflow.DataFlow* method), 746
- to\_yaml() (*ads.jobs.builders.infrastructure.dataflow.DataFlowApp* method), 750
- to\_yaml() (*ads.jobs.builders.infrastructure.dataflow.DataFlowRun* method), 752
- to\_yaml() (*ads.jobs.builders.infrastructure.dsc\_job.DataScienceJobRun* method), 763
- to\_yolo() (*ads.data\_labeling.boundingbox.BoundingBoxItem* method), 543
- to\_yolo() (*ads.data\_labeling.boundingbox.BoundingBoxItems* method), 544
- top\_left (*ads.data\_labeling.boundingbox.BoundingBoxItem* attribute), 542, 543
- top\_right (*ads.data\_labeling.boundingbox.BoundingBoxItem* attribute), 542, 543
- TqdmProgressBar (class in *ads.dataset.progress*), 620
- train() (*ads.automl.driver.AutoML* method), 488
- train() (*ads.automl.provider.AutoMLProvider* method), 490
- train() (*ads.automl.provider.BaselineAutoMLProvider* method), 491

[train\(\)](#) ([ads.automl.provider.OracleAutoMLProvider](#) method), 493  
[train\\_test\\_split\(\)](#) ([ads.dataset.dataset\\_with\\_target.ADSDatasetWithTarget](#) method), 609  
[train\\_validation\\_test\\_split\(\)](#) ([ads.dataset.dataset\\_with\\_target.ADSDatasetWithTarget](#) method), 609  
[training\\_code](#) ([ads.model.runtime.model\\_provenance\\_details.ModelProvenanceDetails](#) attribute), 845  
[training\\_compartment\\_ocid](#) ([ads.model.runtime.model\\_provenance\\_details.ModelProvenanceDetails](#) attribute), 845  
[training\\_conda\\_env](#) ([ads.model.model\\_properties.ModelProperties](#) attribute), 789  
[training\\_conda\\_env](#) ([ads.model.runtime.model\\_provenance\\_details.ModelProvenanceDetails](#) attribute), 845  
[training\\_data](#) ([ads.evaluations.evaluator.ADSEvaluator](#) attribute), 625  
[training\\_env\\_path](#) ([ads.model.runtime.env\\_info.TrainingEnvInfo](#) attribute), 844  
[training\\_env\\_slug](#) ([ads.model.runtime.env\\_info.TrainingEnvInfo](#) attribute), 844  
[training\\_env\\_type](#) ([ads.model.runtime.env\\_info.TrainingEnvInfo](#) attribute), 844  
[training\\_id](#) ([ads.model.model\\_properties.ModelProperties](#) attribute), 789  
[training\\_python\\_version](#) ([ads.model.model\\_properties.ModelProperties](#) attribute), 789  
[training\\_python\\_version](#) ([ads.model.runtime.env\\_info.TrainingEnvInfo](#) attribute), 844  
[training\\_region](#) ([ads.model.runtime.model\\_provenance\\_details.ModelProvenanceDetails](#) attribute), 845  
[training\\_resource\\_id](#) ([ads.model.model\\_properties.ModelProperties](#) attribute), 790  
[training\\_resource\\_ocid](#) ([ads.model.runtime.model\\_provenance\\_details.ModelProvenanceDetails](#) attribute), 845  
[training\\_script\\_path](#) ([ads.model.model\\_properties.ModelProperties](#) attribute), 790  
[TrainingCode](#) (class in [ads.model.runtime.model\\_provenance\\_details](#)), 845  
[TrainingEnvInfo](#) (class in [ads.model.runtime.env\\_info](#)), 844  
[transform\(\)](#) ([ads.automl.provider.AutoMLFeatureSelection](#) method), 488  
[transform\(\)](#) ([ads.automl.provider.AutoMLPreprocessingTransformer](#) method), 489  
[transform\(\)](#) ([ads.automl.provider.BaselineModel](#) method), 492  
[transform\(\)](#) ([ads.common.model.ADSModel](#) method), 521  
[transform\(\)](#) ([ads.dataset.classification\\_dataset.ClassificationDataset](#) method), 590  
[transform\(\)](#) ([ads.dataset.feature\\_engineering\\_transformer.FeatureEngineeringTransformer](#) method), 615  
[transform\(\)](#) ([ads.dataset.label\\_encoder.DataFrameLabelEncoder](#) method), 619  
[transform\(\)](#) ([ads.dataset.recommendation\\_transformer.RecommendationTransformer](#) method), 621  
[transform\(\)](#) ([ads.dataset.recommendation\\_transformer.RecommendationTransformer](#) method), 621  
[TransformerPipeline](#) (class in [ads.dataset.pipeline](#)), 619  
[trial\\_callback\\_model\\_provenance\\_details](#) ([ads.model.runtime.model\\_provenance\\_details.ModelProvenanceDetails](#) ADSTuner property), 724  
[trials](#) ([ads.hpo.search\\_cv.ADSTuner](#) property), 724  
[trials\\_export\(\)](#) ([ads.hpo.search\\_cv.ADSTuner](#) method), 725  
[trials\\_import\(\)](#) ([ads.hpo.search\\_cv.ADSTuner](#) class method), 725  
[trials\\_remaining](#) ([ads.hpo.search\\_cv.ADSTuner](#) property), 726  
[truncate\\_series\\_top\\_n\(\)](#) (in [ads.common.utils](#) module), 538  
[tune\(\)](#) ([ads.hpo.search\\_cv.ADSTuner](#) method), 726  
[txt](#) ([ads.data\\_labeling.visualizer.text\\_visualizer.LabeledTextItem](#) attribute), 573  
[type\\_of\\_target\(\)](#) ([ads.dataset.dataset\\_with\\_target.ADSDatasetWithTarget](#) method), 610  
[TypeAlreadyAdded](#), 633  
[TypeAlreadyRegistered](#), 633  
[TypeNotFound](#), 633  
[TypeNotFound](#), 633

## U

[UniformDistribution](#) (class in [ads.hpo.distributions](#)), 716  
[Unknown](#) (class in [ads.feature\\_engineering.feature\\_type.unknown](#)), 716  
[unregister\(\)](#) ([ads.feature\\_engineering.feature\\_type.handler.feature\\_validator.FeatureValidator](#) method), 706, 708  
[unregister\(\)](#) ([ads.feature\\_engineering.feature\\_type.handler.feature\\_validator.FeatureValidator](#) method), 708, 709  
[unregister\(\)](#) ([ads.feature\\_engineering.feature\\_type.handler.feature\\_validator.FeatureValidator](#) method), 711, 712  
[UNSUPPORTED](#) ([ads.common.utils.ml\\_task\\_types](#) attribute), 536  
[up\\_sample\(\)](#) ([ads.dataset.classification\\_dataset.ClassificationDataset](#) method), 587  
[up\\_sample\(\)](#) (in module [ads.dataset.helper](#)), 618  
[update\(\)](#) ([ads.dataset.progress.DummyProgressBar](#) method), 620  
[update\(\)](#) ([ads.dataset.progress.IpythonProgressBar](#) method), 620

`update()` (`ads.dataset.progress.ProgressBar` method), 620  
`update()` (`ads.dataset.progress.TqdmProgressBar` method), 620  
`update()` (`ads.jobs.builders.infrastructure.dsc_job.DSCJob` method), 755  
`update()` (`ads.model.deployment.model_deployer.ModelDeployer` method), 804  
`update()` (`ads.model.deployment.model_deployment.ModelDeployment` method), 806, 808  
`update()` (`ads.model.generic_model.GenericModel` method), 785  
`update_action()` (`ads.model.generic_model.SummaryStatus` method), 788  
`update_config()` (`ads.dataflow.dataflow.DataFlowRun` method), 583  
`update_config()` (`ads.dataflow.dataflow.RunObserver` method), 583  
`update_deployment()` (`ads.model.generic_model.GenericModel` class method), 785  
`update_deployment()` (`ads.model.generic_model.GenericModel` method), 770  
`update_model()` (`ads.catalog.model.ModelCatalog` method), 496, 499  
`update_notebook_session()` (`ads.catalog.notebook.NotebookCatalog` method), 502  
`update_project()` (`ads.catalog.project.ProjectCatalog` method), 504  
`update_repository()` (in `ads.database.connection` module), 576  
`update_secret()` (`ads.vault.vault.Vault` method), 876  
`update_status()` (`ads.model.generic_model.SummaryStatus` method), 788  
`upload()` (`ads.dataset.factory.DatasetFactory` static method), 615  
`upload_artifact()` (`ads.jobs.builders.infrastructure.dsc_job.DSCJob` method), 755  
`upload_artifact()` (`ads.model.generic_model.GenericModel` method), 770, 786  
`upload_model()` (`ads.catalog.model.ModelCatalog` method), 497, 499  
`url` (`ads.jobs.builders.runtimes.python_runtime.GitPythonRuntime` property), 740  
`url` (`ads.model.deployment.model_deployment.ModelDeployment` attribute), 805  
`use_training` (`ads.evaluations.evaluator.ADSEvaluator.EvaluationMetrics` attribute), 627  
`USER_CUSTOM_PACK` (`ads.model.runtime.env_info.PACK_TYPE` attribute), 844  
`user_name` (`ads.secrets.adb.ADBSecret` attribute), 851  
`user_name` (`ads.secrets.mysqldb.MySQLDBSecret` attribute), 855  
`user_name` (`ads.secrets.oracledb.OracleDBSecret` attribute), 857  
`user_ocid` (`ads.model.runtime.model_provenance_details.ModelProvenanceDetails` attribute), 845  
`v2_4_4` (`ads.dataflow.dataflow.SPARK_VERSION` attribute), 584  
`v3_0_2` (`ads.dataflow.dataflow.SPARK_VERSION` attribute), 584  
`validate()` (`ads.model.runtime.utils.SchemaValidator` method), 846  
`ValidationError`, 610  
`validator` (`ads.feature_engineering.feature_type.address.Address` attribute), 654, 655  
`validator` (`ads.feature_engineering.feature_type.base.FeatureType` attribute), 656  
`validator` (`ads.feature_engineering.feature_type.boolean.Boolean` attribute), 657, 659  
`validator` (`ads.feature_engineering.feature_type.category.Category` attribute), 660, 661  
`validator` (`ads.feature_engineering.feature_type.constant.Constant` attribute), 662, 663  
`validator` (`ads.feature_engineering.feature_type.continuous.Continuous` attribute), 664, 666  
`validator` (`ads.feature_engineering.feature_type.creditcard.CreditCard` attribute), 666, 670  
`validator` (`ads.feature_engineering.feature_type.datetime.DateTime` attribute), 671, 672  
`validator` (`ads.feature_engineering.feature_type.discrete.Discrete` attribute), 673, 675  
`validator` (`ads.feature_engineering.feature_type.document.Document` attribute), 675  
`validator` (`ads.feature_engineering.feature_type.gis.GIS` attribute), 676, 679  
`validator` (`ads.feature_engineering.feature_type.integer.Integer` attribute), 680, 682  
`validator` (`ads.feature_engineering.feature_type.ip_address.IpAddress` attribute), 682, 683  
`validator` (`ads.feature_engineering.feature_type.ip_address_v4.IpAddressV4` attribute), 684, 686  
`validator` (`ads.feature_engineering.feature_type.ip_address_v6.IpAddressV6` attribute), 687, 688  
`validator` (`ads.feature_engineering.feature_type.lat_long.LatLong` attribute), 689, 692  
`validator` (`ads.feature_engineering.feature_type.object.Object` attribute), 693  
`validator` (`ads.feature_engineering.feature_type.ordinal.Ordinal` attribute), 694, 695  
`validator` (`ads.feature_engineering.feature_type.phone_number.PhoneNumber` attribute), 696, 697  
`validator` (`ads.feature_engineering.feature_type.string.String` attribute), 698, 700

[validator\(ads.feature\\_engineering.feature\\_type.text.Text attribute\), 701](#)  
[validator\(ads.feature\\_engineering.feature\\_type.unknown.Unknown attribute\), 702](#)  
[validator\(ads.feature\\_engineering.feature\\_type.zip\\_code.ZipCode attribute\), 703, 705](#)  
[validator\\_registered\(\) \(ads.feature\\_engineering.accessor.mixin.feature\\_types\\_mixin.ADSFeatureTypesMixin method\), 651](#)  
[validator\\_registered\(\) \(ads.feature\\_engineering.feature\\_type\\_manager.FeatureTypeManager class method\), 637](#)  
[validator\\_registered\(\) \(ads.feature\\_engineering.feature\\_type\\_manager.FeatureTypeManager method\), 635](#)  
[ValidatorAlreadyExists, 709](#)  
[ValidatorNotFound, 709](#)  
[ValidatorWithConditionAlreadyExists, 709](#)  
[ValidatorWithConditionNotFound, 709](#)  
[Vault \(class in ads.vault.vault\), 875](#)  
[vault\\_id\(ads.secrets.big\\_data\\_service.BDSSecretKeeper attribute\), 861](#)  
[verify\(\) \(ads.model.framework.automl\\_model.AutoMLModel method\), 816](#)  
[verify\(\) \(ads.model.framework.lightgbm\\_model.LightGBMModel method\), 820](#)  
[verify\(\) \(ads.model.framework.pytorch\\_model.PyTorchModel method\), 825](#)  
[verify\(\) \(ads.model.framework.sklearn\\_model.SklearnModel method\), 830](#)  
[verify\(\) \(ads.model.framework.tensorflow\\_model.TensorFlowModel method\), 836](#)  
[verify\(\) \(ads.model.framework.xgboost\\_model.XGBoostModel method\), 841](#)  
[verify\(\) \(ads.model.generic\\_model.FrameworkSpecificModel method\), 767](#)  
[verify\(\) \(ads.model.generic\\_model.GenericModel method\), 770, 787](#)  
[version\(ads.model.extractor.automl\\_extractor.AutoMLExtractor property\), 791](#)  
[version\(ads.model.extractor.keras\\_extractor.KerasExtractor property\), 797](#)  
[version\(ads.model.extractor.lightgbm\\_extractor.LightgbmExtractor property\), 794](#)  
[version\(ads.model.extractor.pytorch\\_extractor.PyTorchExtractor property\), 800](#)  
[version\(ads.model.extractor.sklearn\\_extractor.SklearnExtractor property\), 796](#)  
[version\(ads.model.extractor.tensorflow\\_extractor.TensorflowExtractor property\), 798](#)  
[version\(ads.model.extractor.xgboost\\_extractor.XgboostExtractor property\), 793](#)  
[version\(ads.model.framework.automl\\_model.AutoMLModel attribute\), 815](#)  
[version\(ads.model.framework.lightgbm\\_model.LightGBMModel attribute\), 820](#)  
[version\(ads.model.framework.pytorch\\_model.PyTorchModel attribute\), 825](#)  
[version\(ads.model.framework.sklearn\\_model.SklearnModel attribute\), 830](#)  
[version\(ads.model.framework.tensorflow\\_model.TensorFlowModel attribute\), 836](#)  
[version\(ads.model.framework.xgboost\\_model.XGBoostModel attribute\), 840](#)  
[version\(ads.model.generic\\_model.GenericModel attribute\), 769](#)  
[version\(\) \(ads.model.extractor.lightgbm\\_extractor.LightgbmExtractor method\), 793](#)  
[version\(\) \(ads.model.extractor.model\\_info\\_extractor.ModelInfoExtractor method\), 794, 795](#)  
[version\(\) \(ads.model.extractor.pytorch\\_extractor.PyTorchExtractor method\), 799](#)  
[version\(\) \(ads.model.extractor.sklearn\\_extractor.SklearnExtractor method\), 796](#)  
[version\(\) \(ads.model.extractor.tensorflow\\_extractor.TensorflowExtractor method\), 798](#)  
[version\(\) \(ads.model.extractor.xgboost\\_extractor.XgboostExtractor method\), 792](#)  
[versionalize\(\) \(ads.dataset.pipeline.TransformerPipeline method\), 619](#)  
[visualize\\_adaptive\\_sampling\\_trials\(\) \(ads.automl.provider.OracleAutoMLProvider method\), 493](#)  
[visualize\\_algorithm\\_selection\\_trials\(\) \(ads.automl.provider.OracleAutoMLProvider method\), 493](#)  
[visualize\\_feature\\_selection\\_trials\(\) \(ads.automl.provider.OracleAutoMLProvider method\), 494](#)  
[visualize\\_transformation\(\) \(in module ads.dataset.helper\), 618](#)  
[visualize\\_transforms\(\) \(ads.common.model.ADSModel method\), 521](#)  
[visualize\\_transforms\(\) \(ads.dataset.dataset\\_with\\_target.ADSDatasetWithTarget method\), 610](#)  
[visualize\\_tuning\\_trials\(\) \(ads.automl.provider.OracleAutoMLProvider method\), 494](#)  
[With\(ads.common.decorator.runtime\\_dependency.OptionalDependency attribute\), 522](#)  
[with\\_image\\_id\(ads.model.runtime.model\\_provenance\\_details.ModelProvenance attribute\), 845](#)  
[wait\(\) \(ads.dataflow.dataflow.RunObserver method\),](#)



584

`wait()` (`ads.hpo.search_cv.ADSTuner` method), 727

`wait()` (`ads.jobs.builders.infrastructure.dataflow.DataFlowRun` method), 752

`wallet_content` (`ads.secrets.adb.ADBSecret` attribute), 851

`wallet_file_name` (`ads.secrets.adb.ADBSecret` attribute), 851

`wallet_location` (`ads.secrets.adb.ADBSecret` attribute), 851

`wallet_secret_ids` (`ads.secrets.adb.ADBSecret` attribute), 851

`warning` (`ads.feature_engineering.feature_type.address.Address` attribute), 653, 655

`warning` (`ads.feature_engineering.feature_type.base.FeatureType` attribute), 656

`warning` (`ads.feature_engineering.feature_type.boolean.Boolean` attribute), 657, 659

`warning` (`ads.feature_engineering.feature_type.category.Category` attribute), 660, 661

`warning` (`ads.feature_engineering.feature_type.constant.Constant` attribute), 662, 663

`warning` (`ads.feature_engineering.feature_type.continuous.Continuous` attribute), 664, 666

`warning` (`ads.feature_engineering.feature_type.creditcard.CreditCard` attribute), 666, 670

`warning` (`ads.feature_engineering.feature_type.datetime.DateTime` attribute), 670, 672

`warning` (`ads.feature_engineering.feature_type.discrete.Discrete` attribute), 673, 675

`warning` (`ads.feature_engineering.feature_type.document.Document` attribute), 675, 676

`warning` (`ads.feature_engineering.feature_type.gis.GIS` attribute), 676, 679

`warning` (`ads.feature_engineering.feature_type.integer.Integer` attribute), 680, 682

`warning` (`ads.feature_engineering.feature_type.ip_address.IpAddress` attribute), 682, 684

`warning` (`ads.feature_engineering.feature_type.ip_address_v4.IpAddressV4` attribute), 684, 686

`warning` (`ads.feature_engineering.feature_type.ip_address_v6.IpAddressV6` attribute), 687, 688

`warning` (`ads.feature_engineering.feature_type.lat_long.LatLong` attribute), 689, 692

`warning` (`ads.feature_engineering.feature_type.object.Object` attribute), 693

`warning` (`ads.feature_engineering.feature_type.ordinal.Ordinal` attribute), 694, 695

`warning` (`ads.feature_engineering.feature_type.phone_number.PhoneNumber` attribute), 696, 697

`warning` (`ads.feature_engineering.feature_type.string.String` attribute), 698, 700

`warning` (`ads.feature_engineering.feature_type.text.Text` attribute), 700, 701

`warning` (`ads.feature_engineering.feature_type.unknown.Unknown` attribute), 702

`warning` (`ads.feature_engineering.feature_type.zip_code.ZipCode` attribute), 703, 705

`warning()` (`ads.feature_engineering.accessor.mixin.eda_mixin.EDAMixin` method), 649

`warning()` (`ads.feature_engineering.accessor.mixin.eda_mixin_series.EDA` method), 650

`warning_registered()` (`ads.feature_engineering.accessor.mixin.feature_types_mixin.AD` method), 651, 652

`warning_registered()` (`ads.feature_engineering.feature_type_manager.FeatureTypeMan` class method), 637

`warning_registered()` (`ads.feature_engineering.feature_type_manager.FeatureTypeMan` method), 635

`WarningAlreadyExists`, 633

`WarningNotFound`, 633

`watch()` (`ads.jobs.builders.infrastructure.dataflow.DataFlowRun` method), 752

`watch()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJobRun` method), 763

`web()` (`ads.dataset.dataset_browser.DatasetBrowser` static method), 603

`WebDatasets` (class in `ads.dataset.dataset_browser`), 605

`with_access_log()` (`ads.model.deployment.model_deployment_properties` method), 810, 811

`with_archive_bucket()` (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` method), 737

`with_archive_uri()` (`ads.jobs.builders.runtimes.python_runtime.DataFl` method), 737

`with_argument()` (`ads.jobs.builders.runtimes.python_runtime.GitPythonF` method), 740

`with_block_storage_size()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 759

`with_category_log()` (`ads.model.deployment.model_deployment_properties.ModelDep` method), 812

`with_compartment_id()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 746

`with_compartment_id()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 759

`with_configuration()` (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` method), 737

`with_configuration()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 747

`with_configuration()`

`(ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime)` (`ads.jobs.ads_job.Job` method), 734  
`with_custom_conda()` (`ads.jobs.builders.runtimes.python_runtime.NotebookRuntime` method), 741  
`with_custom_conda()` (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` method), 735  
`with_custom_conda()` (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` method), 749  
`with_driver_shape()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 747  
`with_driver_shape_config()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 747  
`with_entrypoint()` (`ads.jobs.builders.runtimes.python_runtime.ScriptRuntime` method), 743  
`with_exclude_tag()` (`ads.jobs.builders.runtimes.python_runtime.NotebookRuntime` method), 741  
`with_execute()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 747  
`with_executor_shape()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 747  
`with_executor_shape_config()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 748  
`with_id()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 748  
`with_infrastructure()` (`ads.jobs.ads_job.Job` method), 734  
`with_instance_configuration()` (`ads.model.deployment.model_deployment_properties.ModelDeploymentProperties` method), 810, 812  
`with_job_infrastructure_type()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 759  
`with_job_type()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 760  
`with_language()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 748  
`with_log_group_id()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 760  
`with_log_id()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 760  
`with_logging_configuration()` (`ads.model.deployment.model_deployment_properties.ModelDeploymentProperties` method), 812  
`with_logs_bucket_uri()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 748  
`with_metastore_id()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 749  
`with_notebook()` (`ads.jobs.builders.runtimes.python_runtime.NotebookRuntime` method), 741  
`with_notebook()` (`ads.jobs.builders.runtimes.python_runtime.NotebookRuntime` method), 741  
`with_predict_log()` (`ads.model.deployment.model_deployment_properties.ModelDeploymentProperties` method), 810, 813  
`with_processor()` (`ads.text_dataset.dataset.DataLoader` method), 871  
`with_project_id()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 760  
`with_script()` (`ads.jobs.builders.runtimes.python_runtime.ScriptRuntime` method), 743  
`with_script_bucket()` (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` method), 738  
`with_script_uri()` (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` method), 738  
`with_service_conda()` (`ads.jobs.builders.runtimes.python_runtime.CondaRuntime` method), 736  
`with_service_conda()` (`ads.jobs.builders.runtimes.python_runtime.DataFlowRuntime` method), 739  
`with_shape_config_details()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 761  
`with_shape_name()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 761  
`with_source()` (`ads.jobs.builders.runtimes.python_runtime.GitPythonRuntime` method), 740  
`with_source()` (`ads.jobs.builders.runtimes.python_runtime.ScriptRuntime` method), 743  
`with_spark_version()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 749  
`with_subnet_id()` (`ads.jobs.builders.infrastructure.dsc_job.DataScienceJob` method), 761  
`with_warehouse_bucket_uri()` (`ads.jobs.builders.infrastructure.dataflow.DataFlow` method), 749  
`with_working_dir()` (`ads.jobs.builders.runtimes.python_runtime.PythonRuntime` method), 742  
`WordProcessor` (class in `ads.text_dataset.extractor`), 874  
`workflow_req_id` (`ads.model.deployment.model_deployment.ModelDeploymentProperties` attribute), 805  
`workflow_state_progress`

[\(ads.model.deployment.model\\_deployment.ModelDeployment attribute\), 805](#)  
[workflow\\_steps \(ads.model.deployment.model\\_deployment.ModelDeployment attribute\), 805](#)  
[working\\_dir \(ads.jobs.builders.runtimes.python\\_runtime.PythonRuntime property\), 742](#)  
[wrap\\_lines\(\) \(in module ads.common.utils\), 538](#)  
[write\\_parquet\(\) \(in module ads.dataset.helper\), 618](#)  
[write\\_score\(\) \(in module ads.common.function.fn\\_util\), 529](#)  
[WrongEntityFormat, 571](#)  
[WrongEntityFormatLabelIsEmpty, 549](#)  
[WrongEntityFormatLabelNotString, 549](#)  
[WrongEntityFormatLengthIsNegative, 549](#)  
[WrongEntityFormatLengthNotInteger, 549](#)  
[WrongEntityFormatOffsetIsNegative, 549](#)  
[WrongEntityFormatOffsetNotInteger, 549](#)  
[WrongHandlerMethodSignature, 709](#)

## X

[XgboostExtractor \(class in ads.model.extractor.xgboost\\_extractor\), 792](#)  
[XGBoostModel \(class in ads.model.framework.xgboost\\_model\), 838](#)

## Y

[y\\_pred \(ads.evaluations.statistical\\_metrics.ModelEvaluator attribute\), 631](#)  
[y\\_score \(ads.evaluations.statistical\\_metrics.ModelEvaluator attribute\), 631](#)  
[y\\_true \(ads.evaluations.statistical\\_metrics.ModelEvaluator attribute\), 631](#)  
[YOLO \(ads.data\\_labeling.constants.Formats attribute\), 545](#)

## Z

[zeros\\_handler\(\) \(in module ads.feature\\_engineering.feature\\_type.handler.warnings\), 713](#)  
[zip\\_code \(ads.feature\\_engineering.adsstring.common\\_regex\\_mixin.CommonRegexMixin property\), 653](#)  
[ZipCode \(class in ads.feature\\_engineering.feature\\_type.zip\\_code\), 703](#)