Lookout for Equipment SDK

Michaël HOARAU

Mar 30, 2022

GETTING STARTED

1 Installation, testing and development **1**
- 1.1 Dependencies . . . 1
- 1.2 User installation . . . 1
- 1.3 Development . . . 2

2 User Guide **3**
- 2.1 Introduction . . . 3
- 2.2 Dataset management . . . 3
- 2.3 Model training . . . 4
- 2.4 Evaluating a trained model . . . 5
- 2.5 Scheduler management . . . 7

3 API Documentation **9**
- 3.1 Schema . . . 9
- 3.2 Datasets . . . 10
- 3.3 Models . . . 14
- 3.4 Evaluation . . . 17
- 3.5 Scheduler . . . 19
- 3.6 Plot . . . 24

Python Module Index **29**

Index **31**

CHAPTER

ONE

INSTALLATION, TESTING AND DEVELOPMENT

1.1 Dependencies

lookoutequipment requires:

- python (>= 3.6)
- boto3 (>= 1.17.48)
- markdown
- numpy
- pandas
- pyarrow
- s3fs

To run the examples Matplotlib (>=3.0.0) is required.

1.2 User installation

If you already have a working installation of numpy, boto3, pandas... you can easily install lookoutequipment using `pip`:

```
pip install lookoutequipment
```

You can also get the latest version of lookoutequipment by cloning the repository:

```
git clone https://github.com/aws-samples/amazon-lookout-for-equipment-python-sdk.git
cd lookoutequipment
pip install .
```

1.3 Development

For more information about our contributing guidelines, please refer to the contribute.

CHAPTER

TWO

USER GUIDE

2.1 Introduction

To build and use and anomaly detection model with Amazon Lookout for Equipment, you need to go through the following steps:

1. Preparing your time series dataset and your historical maintenance time ranges
2. Upload your data on Amazon S3
3. Create a dataset and ingest the S3 data into it
4. Train a Lookout for Equipment model
5. Download and post-process the evaluation results
6. Configure and start a scheduler
7. Upload fresh data to Amazon S3
8. Download the inference results generated by the scheduler

The Amazon Lookout for Equipment SDK will help you streamline steps 3 to 8. Some sample datasets are also provided along with some utility functions to tackle steps 1 and 2.

2.2 Dataset management

2.2.1 Creating a dataset

Let's start by loading a sample dataset and uploading it to a location on S3:

```
from lookoutequipment import dataset

root_dir = 'expander-data
bucket = '<<YOUR-BUCKET>>'     # Replace by your bucket name
prefix = '<<YOUR-PREFIX>>/'    # Don't forget the training slash
role_arn = '<<YOUR-ROLE-ARN>>' # An ARN role with access to your S3 data

data = dataset.load_dataset(
    dataset_name='expander',
    target_dir=root_dir
)
dataset.upload_dataset(root_dir, bucket, prefix)
```

From there we are going to instantiate a class that will help us manage our Lookout for Equipment dataset:

```
lookout_dataset = dataset.LookoutEquipmentDataset(
    dataset_name='my_dataset',
    access_role_arn=role_arn,
    component_root_dir=f'{bucket}/{prefix}training-data'
)
```

You will need to specify an ARN for a role that have access to your data on S3.

The following line creates the dataset in the Lookout for Equipment service. If you log into your AWS Console and browse to the Lookout for Equipment datasets list you will see an empty dataset:

```
lookout_dataset.create()
```

2.2.2 Ingesting data into a dataset

This dataset is empty: let's ingest our prepared data (note the trailing slash at the end of the prefix):

```
response = lookout_dataset.ingest_data(bucket, prefix + 'training-data/')
```

The ingestion process will take a few minutes. If you would like to get a feedback from the ingestion process, you can enable a waiter by replacing the previous command by the following:

```
response = lookout_dataset.ingest_data(
    bucket,
    prefix + 'training-data/',
    wait=True,
    sleep_time=60
)
```

2.3 Model training

Once you have ingested some time series data in your dataset, you can train an anomaly detection model:

```
from lookoutequipment import model

lookout_model = model.LookoutEquipmentModel(model_name='my_model', dataset_name='my_
↪dataset')
lookout_model.set_time_periods(data['evaluation_start'],
                               data['evaluation_end'],
                               data['training_start'],
                               data['training_end'])
lookout_model.set_label_data(bucket=bucket,
                             prefix=prefix + 'label-data/',
                             access_role_arn=role_arn)
lookout_model.set_target_sampling_rate(sampling_rate='PT5M')
response = lookout_model.train()
lookout_model.poll_model_training(sleep_time=300)
```

You will see a progress status update every five minutes until the training is successful. With the sample dataset used in this user guide, the training can take up to an hour.

Once your model is trained, you can either check the results over the evaluation period or configure an inference scheduler.

2.4 Evaluating a trained model

2.4.1 Plot detected events

Once a model is trained, the DescribeModel API from Amazon Lookout for Equipment will record the metrics associated to the training.

This API returns a dictionnary with two fields of interest to plot the evaluation results: `labelled_ranges` and `predicted_ranges` which respectively contain the known and predicted anomalies in the evaluation range. Use the following SDK command to get both of these in a Pandas dataframe:

```
from lookoutequipment import evaluation

LookoutDiagnostics = evaluation.LookoutEquipmentAnalysis(model_name='my_model', tags_
↪df=data['data'])
predicted_ranges = LookoutDiagnostics.get_predictions()
labeled_range = LookoutDiagnostics.get_labels()
```

Note: the labeled range from the DescribeModel API, only provides any labelled data falling *within the evaluation range*. If you want to plot or use all of them (including the labels falling within the training range), you should use the original label data by replacing the last line of the previous code snippet by the following code:

```
labels_fname = os.path.join(root_dir, 'labels.csv')
labeled_range = LookoutDiagnostics.get_labels(labels_fname)
```

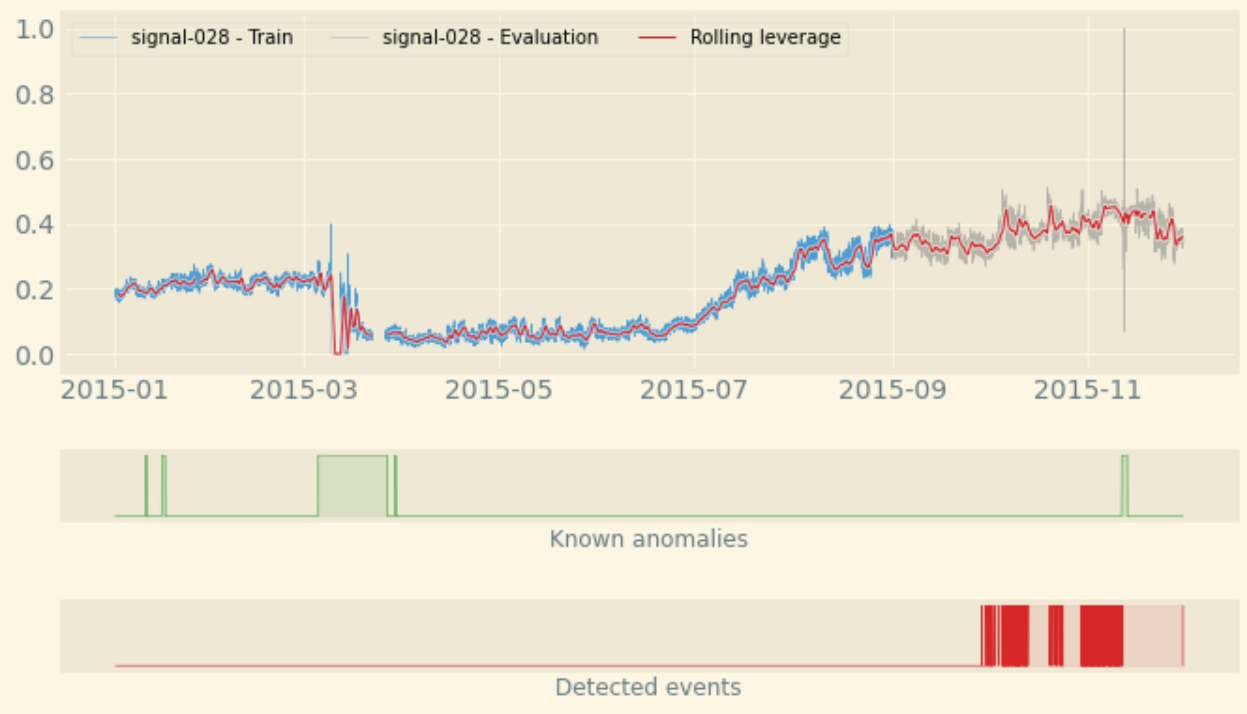
You can then plot one of the original time series signal and add an overlay of the labeled and predicted anomalous events by levering the plot utilities:

```
from lookoutequipment import plot

TSViz = plot.TimeSeriesVisualization(timeseries_df=data['data'], data_format='tabular')
TSViz.add_signal(['signal-001'])
TSViz.add_labels(labeled_range)
TSViz.add_predictions([predicted_ranges])
TSViz.add_train_test_split(data['evaluation_start'])
TSViz.add_rolling_average(60*24)
TSViz.legend_format = {'loc': 'upper left', 'framealpha': 0.4, 'ncol': 3}
fig, axis = TSViz.plot()
```

This code will generate the following plot where you can see:

- A **line plot** for the signal selected: the part used for training the model appears in blue while the evaluation part is in gray.
- The **rolling average** appears as a thin red line overlayed over the time series.
- The **labels** are shown in a green ribbon labelled "Known anomalies" (by default)
- The **predicted events** are shown in a red ribbon labelled "Detected events"



2.4.2 Plot signal distribution

You might be curious about why Amazon Lookout for Equipment detected an anomalous event. Sometime, looking at a few of the time series is enough. But sometime, you need to dig deeper.

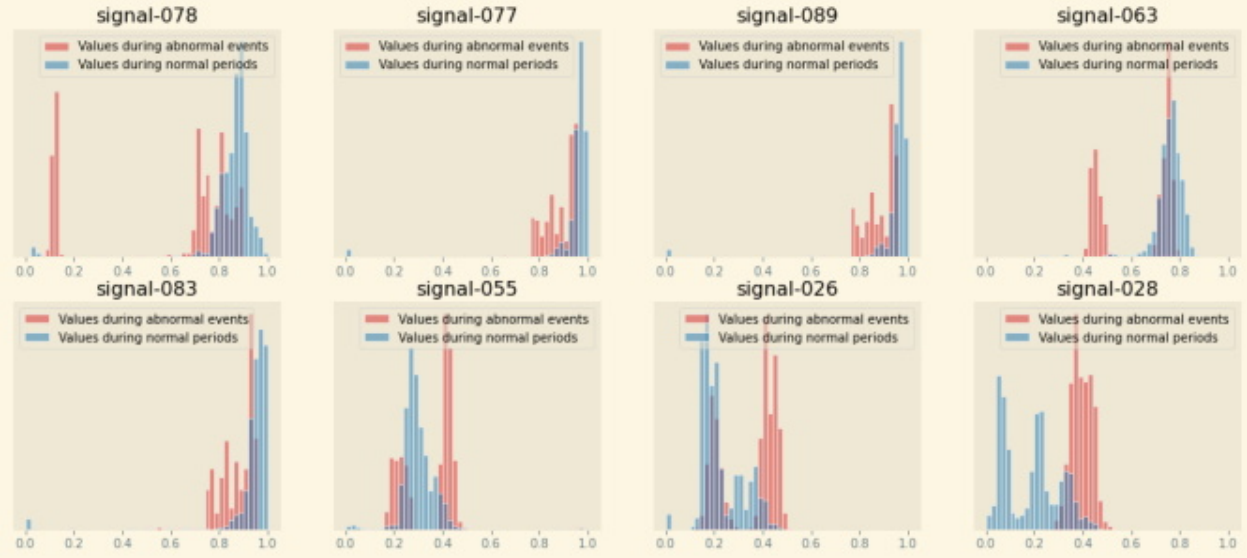
The following function, aggregate the signal importance of every signals over the evaluation period and sum these contributions over time for each signal. Then, it takes the top 8 signals and plot two distributions: one with the values each signal takes during the normal periods (present in the evaluation range) and a second one with the values taken during all the anomalous events detected in the evaluation range. This will help you visualize any significant shift of values for the top contributing signals.

You can also restrict these histograms over a specific range of time by setting the `start` and `end` arguments of the following function with datetime values:

```
from lookoutequipment import plot

TSViz = plot.TimeSeriesVisualization(timeseries_df=data['data'], data_format='tabular')
TSViz.add_predictions([predicted_ranges])
fig = TSViz.plot_histograms(freq='5min')
```

This code will generate the following plot where you can see a histogram for the top 8 signals contributing to the detected events present in the evaluation range of the model:



2.5 Scheduler management

Once a model is successfully trained, you can configure a scheduler that will run regular inferences based on this model:

```
from lookout import scheduler

lookout_scheduler = scheduler.LookoutEquipmentScheduler(
    scheduler_name='my_scheduler',
    model_name='my_model'
)

scheduler_params = {
    'input_bucket': bucket,
    'input_prefix': prefix + 'inference-data/input/',
    'output_bucket': bucket,
    'output_prefix': prefix + 'inference-data/output/',
    'role_arn': role_arn,
    'upload_frequency': 'PT5M',
    'delay_offset': None,
    'timezone_offset': '+00:00',
    'component_delimiter': '_',
    'timestamp_format': 'yyyyMMddHHmmss'
}

lookout_scheduler.set_parameters(**scheduler_params)
```

When the scheduler wakes up, it looks for the appropriate files in the input location configured above. It also opens each file and only keep the data based on their timestamp. Use the following command to prepare some inference data using the sample we have been using throughout this user guide:

```
dataset.prepare_inference_data(
    root_dir='expander-data',
    sample_data_dict=data,
```

(continues on next page)

(continued from previous page)

```
    bucket=bucket,
    prefix=prefix
)
response = lookout_scheduler.create()
```

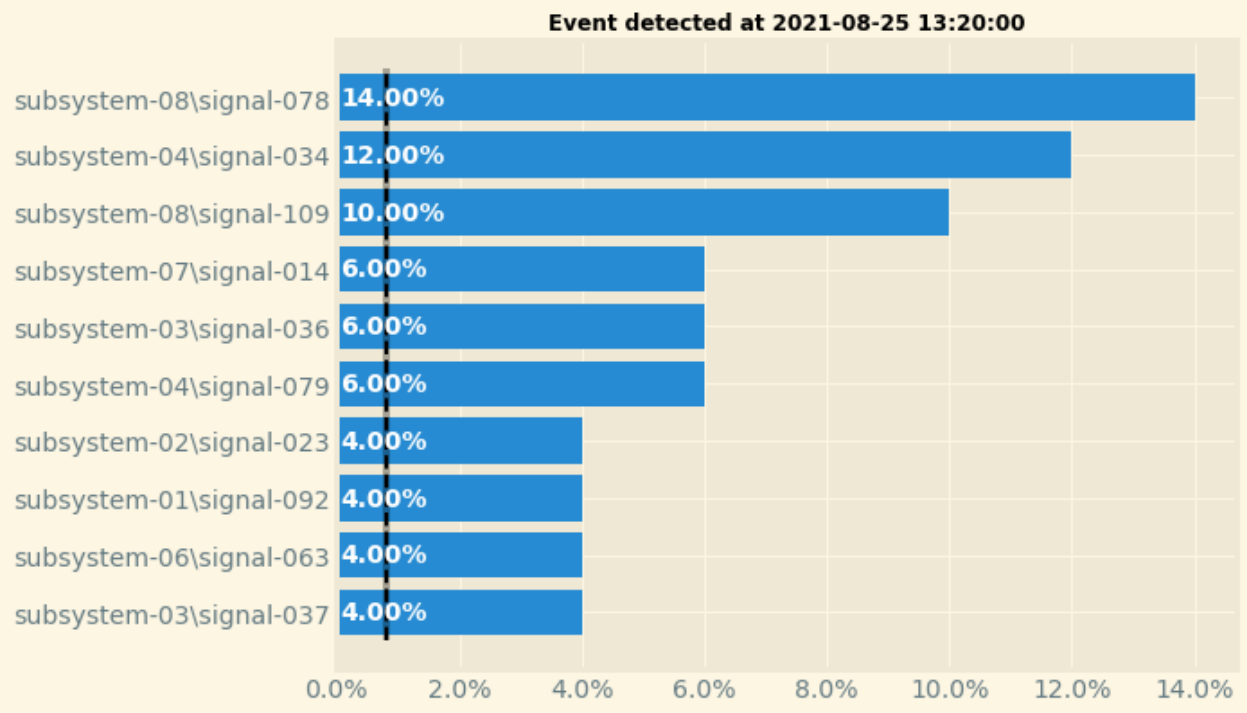
This will create a scheduler that will process one file every 5 minutes (matchin the upload frequency set when configuring the scheduler). After 15 minutes or so, you shoud have some results available. To get these results from the scheduler in a Pandas dataframe, you just have to run the following command:

```
results_df = lookout_scheduler.get_predictions()
```

In this dataframe, you will find one row per event (i.e. one row per scheduler execution). You can then plot the feature importance of any given event. For instance, the following code will plot the feature importance for the first inference execution result:

```
event_details = pd.DataFrame(results_df.iloc[0, 1:]).reset_index()
fig, ax = plot.plot_event_barh(event_details)
```

This is the result you should have with the sample data:



Once you're done, do not forget to stop the scheduler to stop incurring cost:

```
scheduler.stop()
```

You can restart your scheduler with a call to `scheduler.start()` and when you don't have any more use for your scheduler you can delete a stopped scheduler by running `scheduler.delete()`.

CHAPTER

THREE

API DOCUMENTATION

Full API documentation of the *lookoutequipment* Python package.

3.1 Schema

schema.create_data_schema_from_dir(root_dir)	Generates a data schema compatible for Lookout for Equipment from a local directory
schema.create_data_schema_from_s3_path(s3_path)	Generates a data schema compatible for Lookout for Equipment from an S3 directory
schema.create_data_schema(component_fields_map)	Generates a JSON formatted string from a dictionary

3.1.1 create_data_schema_from_dir

src.lookoutequipment.schema.**create_data_schema_from_dir**(*root_dir*)

Generates a data schema compatible for Lookout for Equipment from a local directory

Parameters **root_dir** (*string*) – a path pointing to the root directory where all the CSV files are located

Returns a JSON-formatted string ready to be used as a schema for a Lookout for Equipment dataset

Return type string

3.1.2 create_data_schema_from_s3_path

src.lookoutequipment.schema.**create_data_schema_from_s3_path**(*s3_path*)

Generates a data schema compatible for Lookout for Equipment from an S3 directory

Parameters **s3_path** (*string*) – a path pointing to the root directory on S3 where all the CSV files are located

Returns a JSON-formatted string ready to be used as a schema for a Lookout for Equipment dataset

Return type string

3.1.3 create_data_schema

src.lookoutequipment.schema.**create_data_schema**(*component_fields_map: Dict*)
Generates a JSON formatted string from a dictionary

Parameters **component_fields_map** (*dict*) – a dictionary containing a field maps for the dataset schema

Returns a JSON-formatted string ready to be used as a schema for a dataset

Return type string

3.2 Datasets

dataset.list_datasets([dataset_name_prefix, ...])	List all the Lookout for Equipment datasets available in this account.
dataset.load_dataset(dataset_name, target_dir)	This function can be used to download example datasets to run Amazon Lookout for Equipment on.
dataset.upload_dataset(root_dir, bucket, prefix)	Upload a local dataset to S3.
dataset.prepare_inference_data(root_dir, ...)	This function prepares sequence of data suitable as input for an inference scheduler.
dataset.generate_replay_data(dataset_name, ...)	Generates inference input data from the training data to test a scheduler that would be configured for a model trained with this dataset.
dataset.LookoutEquipmentDataset(...[, ...])	A class to manage Lookout for Equipment datasets

3.2.1 list_datasets

src.lookoutequipment.dataset.**list_datasets**(*dataset_name_prefix=None*, *max_results=50*)
List all the Lookout for Equipment datasets available in this account.

Parameters

- **dataset_name_prefix** (*string*) – prefix to filter out all the datasets which names starts by this prefix. Defaults to None to list all datasets.
- **max_results** (*integer*) – Max number of datasets to return (default: 50)

Returns A list with all the dataset names found in the current region

Return type list of strings

3.2.2 load_dataset

src.lookoutequipment.dataset.**load_dataset**(*dataset_name*, *target_dir*)
This function can be used to download example datasets to run Amazon Lookout for Equipment on.

Parameters

- **dataset_name** (*string*) – Can only be 'expander' at this stage
- **target_dir** (*string*) – Location where to download the data: this location must be readable and writable

Returns dictionnary with data dataframe, labels dataframe, training start and end datetime, evaluation start and end datetime, and the tags description dataframe

Return type data (dict)

3.2.3 upload_dataset

src.lookoutequipment.dataset.**upload_dataset**(*root_dir*, *bucket*, *prefix*)

Upload a local dataset to S3. This method will look for a *training-data* and a *label-data* in the root_dir passed in argument and upload all the content from both these folders to S3.

Parameters

- **root_dir** (*string*) – Path to the local data
- **bucket** (*string*) – Amazon S3 bucket name
- **prefix** (*string*) – Prefix to a directory on Amazon S3 where to upload the data. This prefix *MUST* end with a trailing slash "/"

3.2.4 prepare_inference_data

src.lookoutequipment.dataset.**prepare_inference_data**(*root_dir*, *sample_data_dict*, *bucket*, *prefix*, *num_sequences=3*, *frequency=5*, *start_date=None*)

This function prepares sequence of data suitable as input for an inference scheduler.

Parameters

- **root_dir** (*string*) – Location where the inference data will be written
- **sample_data_dict** (*dict*) – A dictionnary with the sample data as output by *load_dataset()* method
- **bucket** (*string*) – Amazon S3 bucket name
- **prefix** (*string*) – Prefix to a directory on Amazon S3 where to upload the data. This prefix *MUST* end with a trailing slash "/"
- **num_sequences** (*integer*) – Number of short time series sequences to extract: each sequence will be used once by a scheduler. Defaults to 3: a scheduler will run 3 times before failing (unless you provide additional suitable files in the input location)
- **frequency** (*integer*) – The scheduling frequency in minutes: this **MUST** match the resampling rate used to train the model (defaults to 5 minutes)
- **start_date** (*string or datetime*) – The datetime to start the extraction from. Default is None: in this case this method will start looking at date located at the beginning of the evaluation period associated to this sample

3.2.5 generate_replay_data

src.lookoutequipment.dataset.**generate_replay_data**(*dataset_name*, *replay_start_timestamp*, *upload_frequency*, *replay_days=1*, *inference_timezone='UTC'*)

Generates inference input data from the training data to test a scheduler that would be configured for a model trained with this dataset. The data will be output in an S3 location next to your training data S3 location.

Parameters

- **dataset_name** (*string*) – Lookout for Equipment *dataset_name* containing the training data for replaying.
- **replay_start_date** (*string*) – Point in time in the training data from which to begin generating replay data. Example: *"2020-10-01 00:00:00"*
- **upload_frequency** (*string*) – How often replay data is uploaded to the S3 bucket for the inference input data. Valid Values are *PT5M*, *PT10M*, *PT15M*, *PT30M*, or *PT1H*.
- **replay_days** (*integer*) – Duration of the replay data in days (default: 1)
- **inference_timezone** (*string*) – Indicates the timezone for the inference replay dataset. (default: 'UTC')

Returns

(boolean) True if no problem detected, otherwise a list of sequences that could not be generated (which will trigger a failed scheduler execution)

3.2.6 LookoutEquipmentDataset

class src.lookoutequipment.dataset.**LookoutEquipmentDataset**(*dataset_name*, *access_role_arn*, *component_fields_map=None*, *component_root_dir=None*)

A class to manage Lookout for Equipment datasets

Attributes

components_list	list of components part of the schema of this dataset
dataset_name	string with the name given to the dataset
dataset_schema	string with a JSON-formatted string describing the data schema the dataset must conform to
ingestion_job_id	string the ID of the data ingestion job
ingestion_job_response	string with a JSON-formatted string describing the response details of a data ingestion job.
role_arn	string containing the role ARN necessary to access the S3 location where the datasets are stored
schema	dict dictionnary containing the schema of this dataset if it was already created in Lookout for Equipment

Methods

__init__(dataset_name, access_role_arn[, ...])	Create a new instance to configure all the attributes necessary to manage a Lookout for Equipment dataset.
create()	Creates a Lookout for Equipment dataset
delete([force_delete])	Deletes the dataset
get_component_field_map(component)	
ingest_data(bucket, prefix[, wait, sleep_time])	Ingest data from an S3 location into the dataset
list_models()	List all the models trained with this dataset
poll_data_ingestion([sleep_time])	This function polls the data ingestion describe API and prints a status until the ingestion is done.

__init__(*dataset_name*, *access_role_arn*, *component_fields_map=None*, *component_root_dir=None*)
Create a new instance to configure all the attributes necessary to manage a Lookout for Equipment dataset.

Parameters

- **dataset_name** (*string*) – the name of the dataset to manage
- **component_fields_map** (*string*) – the mapping of the different fields associated to this dataset. Either `component_root_dir` or `component_fields_map` must be provided. Defaults to None.
- **component_root_dir** (*string*) – the root location where the sensor data are stored. Either `component_root_dir` or `component_fields_map` must be provided. Defaults to None. Can be a local folder or an S3 location.
- **access_role_arn** (*string*) – the ARN of a role that will allow Lookout for Equipment to read data from the data source bucket on S3

create()
Creates a Lookout for Equipment dataset

Returns Response of the create dataset API

Return type string

delete(*force_delete=True*)
Deletes the dataset

Parameters **force_delete** (*boolean*) – if set to True, also delete all the models that are using this dataset before deleting it. Otherwise, this method will list the attached models (Default: True)

get_component_field_map(*component*)

ingest_data(*bucket*, *prefix*, *wait=False*, *sleep_time=60*)
Ingest data from an S3 location into the dataset

Parameters

- **bucket** (*string*) – Bucket name where the data to ingest are located
- **prefix** (*string*) – Actual location inside the aforementioned bucket
- **wait** (*Boolean*) – If True, this function will wait for the ingestion to finish (default to False)

- **sleep_time** (*integer*) – how many seconds should we wait before polling again when the `wait` parameter is True (default: 60)

Returns Response of the start ingestion job API call (if `wait` is False) or of the actual finished ingestion job (if `wait` is True)

Return type string

list_models()
List all the models trained with this dataset

Returns A list with the names of every models trained with this dataset

Return type list of strings

poll_data_ingestion(*sleep_time=60*)
This function polls the data ingestion describe API and prints a status until the ingestion is done.

Parameters **sleep_time** (*integer*) – How many seconds should we wait before polling again (default: 60)

3.3 Models

model.list_models([model_name_prefix, ...])	List all the models available in the current account
model.LookoutEquipmentModel(model_name, ...)	A class to manage Lookout for Equipment models

3.3.1 list_models

src.lookoutequipment.model.**list_models**(*model_name_prefix=None*, *dataset_name_prefix=None*, *max_results=50*)
List all the models available in the current account

Parameters

- **model_name_prefix** (*string*) – Prefix to filter on the model name to look for (default: None)
- **dataset_name_prefix** (*string*) – Prefix to filter the dataset name: if used, only models making use of this particular dataset are returned (default: None)
- **max_results** (*integer*) – Max number of datasets to return (default: 50)

Returns List of all the models corresponding to the input parameters (regions and dataset)

Return type List of strings

3.3.2 LookoutEquipmentModel

class src.lookoutequipment.model.**LookoutEquipmentModel**(*model_name*, *dataset_name*)
A class to manage Lookout for Equipment models

dataset_name
The name of the dataset used to train the model attached to a given class instance

Type string

model_name
The name of the model attached to a given class instance

Type string

create_model_request
The parameters to be used to train the model

Type dict

Methods

__init__(model_name, dataset_name)	Create a new instance to configure all the attributes necessary to manage a Lookout for Equipment model.
delete()	Delete the current model
poll_model_training([sleep_time])	This function polls the model describe API and prints a status until the training is done.
set_label_data(bucket, prefix, access_role_arn)	Tell Lookout for Equipment to look for labelled data to train the model and where to find them on S3
set_off_condition(off_condition)	Configure off-time detection using one of your machine's sensors.
set_off_conditions(off_conditions_string)	Tells Lookout for Equipment to use one of the signals as a guide to tell if the asset/process is currently on or off.
set_subset_schema(field_map)	Configure the inline data schema that will let Lookout for Equipment knows that it needs to select a subset of all the signals configured at ingestion
set_target_sampling_rate(sampling_rate)	Set the sampling rate to use before training the model
set_time_periods(evaluation_start, ...)	Set the training / evaluation time split
train()	Train the model as configured with this object

__init__(*model_name*, *dataset_name*)
Create a new instance to configure all the attributes necessary to manage a Lookout for Equipment model.

Parameters

- **model_name** (*string*) – the name of the model to manage
- **dataset_name** (*string*) – the name of the dataset associated to the model

delete()
Delete the current model

Returns The delete model API response in JSON format

Return type string

poll_model_training(*sleep_time=60*)
This function polls the model describe API and prints a status until the training is done.

Parameters **sleep_time** (*integer*) – How many seconds should we wait before polling again (default: 60)

set_label_data(*bucket*, *prefix*, *access_role_arn*)
Tell Lookout for Equipment to look for labelled data to train the model and where to find them on S3

Parameters

- **bucket** (*string*) – Bucket name where the labelled data can be found
- **prefix** (*string*) – Prefix where the labelled data can be found
- **access_role_arn** (*string*) – A role that Lookout for Equipment can use to access the bucket and prefix aforementioned

set_off_condition(*off_condition*)
Configure off-time detection using one of your machine's sensors.

Parameters **off_condition** (*string*) – Sensor representative of the machine's on/off state. Ex: 'tag_name < 1000'

set_off_conditions(*off_conditions_string*)
Tells Lookout for Equipment to use one of the signals as a guide to tell if the asset/process is currently on or off.

Parameters **off_conditions_string** (*string*) – A string with the format *component_nametag_name>0.0* where the condition can either be < or > with a real value materializing the boundary used to identify off time from on time.

set_subset_schema(*field_map*)
Configure the inline data schema that will let Lookout for Equipment knows that it needs to select a subset of all the signals configured at ingestion

Parameters **field_map** – string A JSON string describing which signals to keep for this model

set_target_sampling_rate(*sampling_rate*)
Set the sampling rate to use before training the model

Parameters **sampling_rate** (*string*) – One of [PT1M, PT5S, PT15M, PT1S, PT10M, PT15S, PT30M, PT10S, PT30S, PT1H, PT5M]

set_time_periods(*evaluation_start*, *evaluation_end*, *training_start*, *training_end*)
Set the training / evaluation time split

Parameters

- **evaluation_start** (*datetime*) – Start of the evaluation period
- **evaluation_end** (*datetime*) – End of the evaluation period
- **training_start** (*datetime*) – Start of the training period
- **training_end** (*datetime*) – End of the training period

train()
Train the model as configured with this object

Returns The create model API response in JSON format

Return type string

3.4 Evaluation

evaluation.LookoutEquipmentAnalysis(...)	A class to manage Lookout for Equipment result analysis

3.4.1 LookoutEquipmentAnalysis

class src.lookoutequipment.evaluation.**LookoutEquipmentAnalysis**(*model_name*, *tags_df*)

A class to manage Lookout for Equipment result analysis

model_name
the name of the Lookout for Equipment trained model

Type string

predicted_ranges
a Pandas dataframe with the predicted anomaly ranges listed in chronological order with a Start and End columns

Type pandas.DataFrame

labelled_ranges
A Pandas dataframe with the labelled anomaly ranges listed in chronological order with a Start and End columns

Type pandas.DataFrame

df_list
A list with each time series into a dataframe

Type list of pandas.DataFrame

Methods

__init__(model_name, tags_df)	Create a new analysis for a Lookout for Equipment model.
compute_histograms([index_normal, ...])	This method loops through each signal and computes two distributions of the values in the time series: one for all the anomalies found in the evaluation period and another one with all the normal values found in the same period.
get_labels([labels_fname])	Get the labelled ranges as provided to the model before training
get_predictions()	Get the anomaly ranges predicted by the current model
get_ranked_list([max_signals])	Returns the list of signals with computed rank.
plot_histograms([nb_cols, max_plots])	Once the histograms are computed, we can plot the top N by decreasing ranking distance.
plot_histograms_v2(custom_ranking[, ...])	
plot_signals([nb_cols, max_plots])	Once the histograms are computed, we can plot the top N signals by decreasing ranking distance.
set_time_periods(evaluation_start, ...)	Set the time period of analysis

__init__(*model_name*, *tags_df*)
Create a new analysis for a Lookout for Equipment model.

Parameters

- **model_name** (*string*) – The name of the Lookout for Equipment trained model
- **tags_df** (*pandas.DataFrame*) – A dataframe containing all the signals, indexed by time
- **region_name** (*string*) – Name of the AWS region from where the service is called.

compute_histograms(*index_normal=None*, *index_anomaly=None*, *num_bins=20*)
This method loops through each signal and computes two distributions of the values in the time series: one for all the anomalies found in the evaluation period and another one with all the normal values found in the same period. It then computes the Wasserstein distance between these two histograms and then rank every signals based on this distance. The higher the distance, the more different a signal is when comparing anomalous and normal periods. This can orient the investigation of a subject matter expert towards the sensors and associated components.

Parameters

- **index_normal** (*pandas.DateTimeIndex*) – All the normal indices
- **index_anomaly** (*pandas.DateTimeIndex*) – All the indices for anomalies
- **num_bins** (*integer*) – Number of bins to use to build the distributions (default: 20)

get_labels(*labels_fname=None*)
Get the labelled ranges as provided to the model before training

Parameters **labels_fname** (*string*) – As an option, if you provide a path to a CSV file containing the label ranges, this method will use this file to load the labels. If this argument is not provided, it will load the labels from the trained model Describe API (Default to None)

Returns A Pandas dataframe with the labelled anomaly ranges listed in chronological order with a Start and End columns

Return type pandas.DataFrame

get_predictions()
Get the anomaly ranges predicted by the current model

Returns A Pandas dataframe with the predicted anomaly ranges listed in chronological order with a Start and End columns

Return type pandas.DataFrame

get_ranked_list(*max_signals=12*)
Returns the list of signals with computed rank.

Parameters **max_signals** (*integer*) – Number of signals to consider (default: 12)

Returns A dataframe with each signal and the associated rank value

Return type pandas.DataFrame

plot_histograms(*nb_cols=3*, *max_plots=12*)
Once the histograms are computed, we can plot the top N by decreasing ranking distance. By default, this will plot the histograms for the top 12 signals, with 3 plots per line.

Parameters

- **nb_cols** (*integer*) – Number of plots to assemble on a given row (default: 3)
- **max_plots** (*integer*) – Number of signal to consider (default: 12)

Returns

tuple containing:

- A `matplotlib.pyplot.figure` where the plots are drawn
- A `list of matplotlib.pyplot.Axis` with each plot drawn here

Return type tuple

plot_histograms_v2(*custom_ranking*, *nb_cols=3*, *max_plots=12*, *num_bins=20*)

plot_signals(*nb_cols=3*, *max_plots=12*)

Once the histograms are computed, we can plot the top N signals by decreasing ranking distance. By default, this will plot the signals for the top 12 signals, with 3 plots per line. For each signal, this method will plot the normal values in green and the anomalies in red.

Parameters

- **nb_cols** (*integer*) – Number of plots to assemble on a given row (default: 3)
- **max_plots** (*integer*) – Number of signal to consider (default: 12)

Returns

tuple containing:

- A `matplotlib.pyplot.figure` where the plots are drawn
- A `list of matplotlib.pyplot.Axis` with each plot drawn here

Return type tuple

set_time_periods(*evaluation_start*, *evaluation_end*, *training_start*, *training_end*)

Set the time period of analysis

Parameters

- **evaluation_start** (*datetime*) – Start of the evaluation period
- **evaluation_end** (*datetime*) – End of the evaluation period
- **training_start** (*datetime*) – Start of the training period
- **training_end** (*datetime*) – End of the training period

3.5 Scheduler

scheduler.LookoutEquipmentScheduler(...)	A class to represent a Lookout for Equipment inference scheduler object.
scheduler.LookoutEquipmentSchedulerInspector(A ...	class to be used to inspect existing inference scheduler and output a report about how the inputs should be structured

3.5.1 LookoutEquipmentScheduler

class src.lookoutequipment.scheduler.**LookoutEquipmentScheduler**(*scheduler_name*, *model_name*)

A class to represent a Lookout for Equipment inference scheduler object.

scheduler_name
Name of the scheduler associated to this object

Type string

model_name
Name of the model used to run the inference when the scheduler wakes up

Type string

create_request
A dictionary containing all the parameters to configure and create an inference scheduler

Type dict

execution_summaries
A list of all inference execution results. Each execution is stored as a dictionary.

Type list of dict

Methods

__init__(scheduler_name, model_name)	Constructs all the necessary attributes for a scheduler object.
create([wait])	Create an inference scheduler for a trained Lookout for Equipment model
delete()	Delete the current inference scheduler
get_predictions()	This method loops through all the inference executions and build a dataframe with all the predictions generated by the model
get_status()	Get current status of the inference scheduler
list_inference_executions([...])	This method lists all the past inference execution triggered by the current scheduler.
set_parameters(input_bucket, input_prefix, ...)	Set all the attributes for the scheduler object.
start([wait])	Start an existing inference scheduler if it exists
stop([wait])	Stop an existing started inference scheduler

__init__(*scheduler_name*, *model_name*)
Constructs all the necessary attributes for a scheduler object.

Parameters

- **scheduler_name** (*string*) – The name of the scheduler to be created or managed
- **model_name** (*string*) – The name of the model to schedule inference for
- **region_name** (*string*) – Name of the AWS region from where the service is called.

create(*wait=True*)
Create an inference scheduler for a trained Lookout for Equipment model

Parameters **wait** (*boolean*) – Wait for the creation process to finish (default: True)

delete()
Delete the current inference scheduler

Returns A JSON dictionary with the response from the delete request API

Return type dict

get_predictions()
This method loops through all the inference executions and build a dataframe with all the predictions generated by the model

Returns A dataframe with one prediction by row (1 for an anomaly or 0 otherwise). Each row is indexed by timestamp.

Return type pandas.DataFrame

get_status()
Get current status of the inference scheduler

Returns The status of the inference scheduler, as extracted from the DescribeInferenceScheduler API

Return type string

list_inference_executions(*execution_status=None*, *start_time=None*, *end_time=None*, *max_results=50*)
This method lists all the past inference execution triggered by the current scheduler.

Parameters

- **execution_status** (*string*) – Only keep the executions with a given status (default: None)
- **start_time** (*pandas.DateTime*) – Filters out the executions that happened before start_time (default: None)
- **end_time** (*pandas.DateTime*) – Filters out the executions that happened after end_time (default: None)
- **max_results** (*integer*) – Max number of results you want to get out of this method (default: 50)

Returns A list of all past inference executions, with each inference attributes stored in a python dictionary

Return type list of dict

set_parameters(*input_bucket*, *input_prefix*, *output_bucket*, *output_prefix*, *role_arn*, *upload_frequency='PT5M'*, *delay_offset=None*, *timezone_offset='+00:00'*, *component_delimiter='_'*, *timestamp_format='yyyyMMddHHmmss'*)
Set all the attributes for the scheduler object.

Parameters

- **input_bucket** (*string*) – Bucket when the input data are located
- **input_prefix** (*string*) – Location prefix for the input data
- **output_bucket** (*string*) – Bucket location for the inference execution output
- **output_prefix** (*string*) – Location prefix for the inference result file
- **role_arn** (*string*) – Role allowing Lookout for Equipment to read and write data from the input and output bucket locations
- **upload_frequency** (*string*) – Upload frequency of the data (default: PT5M)

- **delay_offset** (*integer*) – Offset in minute, ensuring the data are available when the scheduler wakes up to run the inference (default: None)
- **timezone_offset** (*string*) – Timezone offset used to match the date in the input filenames (default: +00:00)
- **component_delimiter** (*string*) – Character to use to delimit component name and timestamp in the input filenames (default: '_')
- **timestamp_format** (*string*) – Format of the timestamp to look for in the input filenames (default: yyyyMMddHHmmss)

start(*wait=True*)

Start an existing inference scheduler if it exists

Parameters **wait** (*boolean*) – Wait for the starting process to finish (default: True)

stop(*wait=True*)

Stop an existing started inference scheduler

Parameters **wait** (*boolean*) – Wait for the stopping process to finish (default: True)

3.5.2 LookoutEquipmentSchedulerInspector

class src.lookoutequipment.scheduler.**LookoutEquipmentSchedulerInspector**(*scheduler_name*, *current_timezone='UTC'*)

A class to be used to inspect existing inference scheduler and output a report about how the inputs should be structured

scheduler_name

Name of the scheduler to inspect

Type string

delay_offset

Data delay offset of the scheduler

Type integer

frequency

Data upload frequency of the scheduler

Type integer

timestamp_format

Timestamp format expected by the scheduler

Type string

delimiter

Delimiter character between component and timestamp in the name of the input CSV file

Type string

timezone

In which timezone are you located?

Type string

input_timezone_offset

Timezone offset between current location and data

Type string

s3_input_location
Location of the input CSV file to run inference on

Type string

current_time
Current time at which the inspection report is generated

Type datetime

start_time
Start time the scheduler looks for when looking for valid data points to run inference on

Type datetime

end_time
End time the scheduler looks for when looking for valid data points to run inference on

Type datetime

next_wakeup_time
Computed next time the scheduler will wake up

Type datetime

next_timestamp
Next timestamp the scheduler will look for in the input file names

Type datetime

Methods

__init__(scheduler_name[, current_timezone])	A class to inspect an existing inference scheduler and output a report in either Markdown (to be printed in a Jupyter notebook for instance) or in an independant HTML file
build_inspection_report()	Build the inspection report in Markdown format suitable for displaying from a Jupyter notebook or any output that can take this format input
export_to_html([html_path])	This method will export the Markdown report as built by the `build_inspection_report()` method into an HTML file.
get_next_time_range()	Get the current time and derives the next time the scheduler will wake up, what timestamp it will look for to find the right input file to process and which time range to filter out when opening this file.

__init__(*scheduler_name*, *current_timezone='UTC'*)
A class to inspect an existing inference scheduler and output a report in either Markdown (to be printed in a Jupyter notebook for instance) or in an independant HTML file

Parameters

- **scheduler_name** (*string*) – Name of the scheduler to inspect. This inference scheduler must already exist
- **current_timezone** (*string*) – Timezone where the data are generated. Must be one of the timezone referenced in the `pytz.all_timezones` list

Raises **Exception** – if a scheduler with this name is not found

build_inspection_report()
Build the inspection report in Markdown format suitable for displaying from a Jupyter notebook or any output that can take this format input

Returns The inspection report in Markdown format

Return type string

export_to_html(*html_path=None*)
This method will export the Markdown report as built by the `build_inspection_report()` method into an HTML file.

Parameters **html_path** (*string*) – path + filename location to write the HTML report to. By default, will write the file to the current location, using the scheduler name and suffixing it with '-inspection-report' (default: None)

get_next_time_range()
Get the current time and derives the next time the scheduler will wake up, what timestamp it will look for to find the right input file to process and which time range to filter out when opening this file.

3.6 Plot

plot.plot_histogram_comparison(timeseries_1, ...)	Takes two timeseries and plot a histogram showing their respective distribution of values
plot.plot_event_barh(event_details[, ...])	Plot a horizontal bar chart with the feature importance of each signal that contributes to the event passed as an argument.
plot.plot_range(range_df, range_title, ...)	Plot a range with either labelled or predicted events as a filled area positionned under the timeseries data.
plot.TimeSeriesVisualization(timeseries_df, ...)	A class to manage time series visualization along with labels and detected events

3.6.1 plot_histogram_comparison

src.lookoutequipment.plot.**plot_histogram_comparison**(*timeseries_1*, *timeseries_2*, *ax=None*, *label_timeseries_1=None*, *label_timeseries_2=None*, *show_legend=True*, *num_bins=10*)
Takes two timeseries and plot a histogram showing their respective distribution of values

Parameters

- **timeseries_1** (*array_like*) – The first time series to plot a histogram for
- **timeseries_2** (*array_like*) – The second time series to plot a histogram for
- **ax** (*matplotlib.pyplot.Axis*) – The ax in which to render the range plot. If None, this function will create a figure and an ax. Default to None
- **label_timeseries_1** (*string*) – The label for the first time series
- **label_timeseries_2** (*string*) – The label for the second time series
- **show_legend** (*Boolean*) – True to display a legend on this histogram plot and False otherwise
- **num_bins** (*integer*) – Number of bins to compute (defaults to 10)

3.6.2 plot_event_barh

src.lookoutequipment.plot.**plot_event_barh**(*event_details*, *num_signals=10*, *fig_width=12*)

Plot a horizontal bar chart with the feature importance of each signal that contributes to the event passed as an argument.

Parameters

- **event_details** (*pandas.DataFrame*) – A dataframe with the sensor name and the feature importance in two columns
- **num_signals** (*integer*) – States how many signals to plot in the bar chart (default to 10)
- **fig_width** (*integer*) – Width of the figure to plot

Returns

tuple: tuple containing:

- A `matplotlib.pyplot.figure` where the plot is drawn
- A `matplotlib.pyplot.Axis` where the plot is drawn

Return type Returns

3.6.3 plot_range

src.lookoutequipment.plot.**plot_range**(*range_df*, *range_title*, *color*, *ax*, *column_name*)

Plot a range with either labelled or predicted events as a filled area positionned under the timeseries data.

Parameters

- **range_df** (*pandas.DataFrame*) – A DataFrame that must contain at least a DateTimeIndex and a column called "Label"
- **range_title** (*string*) – Title of the ax containing this range
- **color** (*string*) – A string used as a color for the filled area of the plot
- **ax** (*matplotlib.pyplot.Axis*) – The ax in which to render the range plot
- **column_name** (*string*) – The column from the range_df dataframe to use to plot the range

3.6.4 TimeSeriesVisualization

class src.lookoutequipment.plot.**TimeSeriesVisualization**(*timeseries_df*, *data_format*, *timestamp_col=None*, *tag_col=None*, *resample=None*, *verbose=False*)

A class to manage time series visualization along with labels and detected events

Attributes

DEFAULT_COLORS	
data	A pandas.DataFrame containing time series data to plot
format	Either timeseries or tabular depending on the format of your time series.
legend_format	kwargs dict to configure the legend to be displayed when this class renders the requested plot
signal_data	list of pandas.DataFrame containing the time series data to plot
tag_col	If *data_format* is timeseries, this argument specifies what is the name of the columns that contains the name of the tags
tags_list	list of strings containing the list of all tags associated to the current dataset
timestamp_col	string specifying the name of the columns that contains the timestamps

Methods

__init__(timeseries_df, data_format[, ...])	Create a new instance to plot time series with different data structure
add_labels(labels_df[, labels_title])	Add a label component to the plot to visualize the known anomalies periods as a secondary plot under the time series visualization panel.
add_predictions(predictions_list[, ...])	Add a prediction component to the plot to visualize detected events as a secondary plot under the time series visualization panel.
add_rolling_average(window_size)	Adds a rolling average over a time series plot
add_signal(signals_list)	This method will let you select which signals you want to plot.
add_train_test_split(split_timestamp[, ...])	Add a way to visualize the split between training and testing periods.
plot([fig_width, colors, labels_bottom, ...])	Renders the plot as configured with the previous function
plot_histograms([freq, prediction_index, ...])	Plot values distribution as histograms for the top contributing sensors.

__init__(*timeseries_df*, *data_format*, *timestamp_col=None*, *tag_col=None*, *resample=None*, *verbose=False*)

Create a new instance to plot time series with different data structure

Parameters

- **timeseries_df** (*pandas.DataFrame*) – A dataframe containing time series data that you want to plot
- **data_format** (*string*) – Use "timeseries" if your dataframe has three columns: timestamp, values and tagname. Use "tabular" if timestamp is your first column and all the other tags are in the following columns: timestamp, tag1, tag2...

- **timestamp_col** (*string*) – Specifies the name of the columns that contains the timestamps. If set to None, it means the timestamp is already an index (default to None)
- **tag_col** (*string*) – If data_format is "timeseries", this argument specifies what is the name of the columns that contains the name of the tags
- **resample** (*string*) – If specified, this class will resample the data before plotting them. Use the same format than the string rule as used in the `pandas.DataFrame.resample()` method (default to None)
- **verbose** (*boolean*) – If True, this class will print some messages along the way (defaults to False)

add_labels(*labels_df*, *labels_title='Known anomalies'*)

Add a label component to the plot to visualize the known anomalies periods as a secondary plot under the time series visualization panel.

Parameters

- **labels_df** (*pandas.DataFrame*) – You can add one label ribbon, defined with a dataframe that gives the start and end date of every known anomalies
- **labels_title** (*string*) – Title to be used for the known anomalies label ribbon

add_predictions(*predictions_list*, *prediction_titles=['Detected events']*)

Add a prediction component to the plot to visualize detected events as a secondary plot under the time series visualization panel.

Parameters

- **predictions_list** (*list of pandas.DataFrame*) – You can add several predictions ribbon. Each ribbon is defined with a dataframe that gives the start and end date of every detected events. Several ribbons can be grouped inside a list
- **prediction_titles** (*list of strings*) – This lists contains all the titles to be used for each prediction ribbon

add_rolling_average(*window_size*)

Adds a rolling average over a time series plot

Parameters **window_size** (*integer*) – Size of the window in time steps to average over

add_signal(*signals_list*)

This method will let you select which signals you want to plot. It will double check that the signals are, actually available in the tags list. This method will populate the `signal_data` property with the list of each dataframes containing the signals to plot.

Parameters **signals_list** (*list of string*) – A list of tag names to be rendered when you call `plot()`

Raises **Exception** – if some of the signals are not found in the tags list

add_train_test_split(*split_timestamp*, *train_label='Train'*, *test_label='Evaluation'*)

Add a way to visualize the split between training and testing periods. The training period will stay colorful on the timeseries area of the plot while the testing period will be greyed out.

Parameters

- **split_timestamp** (*string or datetime*) – The split date. If a string is passed, it will be converted into a datetime
- **train_label** (*string*) – Name of the training period (will be visible in the legend)
- **test_label** (*string*) – Name of the testing period (will be visible in the legend)

plot(*fig_width=18*, *colors={'labels': 'tab:green', 'predictions': 'tab:red'}*, *labels_bottom=False*, *no_legend=False*)
Renders the plot as configured with the previous function

Parameters **fig_width** (*integer*) – The width of the figure to generate (defaults to 18)

Returns

tuple containing:

- A `matplotlib.pyplot.figure` where the plots are drawn
- A `list of matplotlib.pyplot.Axis` with each plot drawn here

Return type tuple

plot_histograms(*freq='1min'*, *prediction_index=0*, *top_n=8*, *fig_width=18*, *start=None*, *end=None*)
Plot values distribution as histograms for the top contributing sensors.

Parameters

- **freq** (*string*) – The datetime index frequence (defaults to '1min'). This must be a string following this format: XXmin where XX is a number of minutes.
- **prediction_index** (*integer*) – You can add several predicted ranges in your plot. Use this argument to specify for which one you wish to plot a histogram for (defaults to 0)
- **top_n** (*integer*) – Number of top signals to plot (default: 8)
- **fig_width** (*float*) – Width of the figure generated (default: 18)
- **start** (*pandas.DatetTime*) – Start date of the range to build the values distribution for (default: None, use the evaluation period start)
- **end** (*pandas.DatetTime*) – End date of the range to build the values distribution for (default: None, use the evaluation period end)

Returns a figure where the histograms are drawn

Return type matplotlib.pyplot.figure

The **Amazon Lookout for Equipment SDK** is an open source Python package that allows data scientists and software developers to easily build and deploy time series anomalie detection models using Amazon Lookout for Equipment. This SDK enables to do the following:

- Build dataset schema
- Data upload to the necessary S3 structure
- Train an anomaly detection model using Amazon Lookout for Equipment
- Build beautiful visualization for your model evaluation
- Configure and start an inference scheduler
- Manage schedulers (start, stop, delete) whenever necessary
- Visualize scheduler inferences results

PYTHON MODULE INDEX

S

src.lookoutequipment.dataset, 10
src.lookoutequipment.evaluation, 17
src.lookoutequipment.model, 14
src.lookoutequipment.plot, 24
src.lookoutequipment.scheduler, 19
src.lookoutequipment.schema, 9

INDEX

Symbols

__init__() (*src.lookoutequipment.dataset.LookoutEquipmentDataset method*), 13
__init__() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 17
__init__() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 15
__init__() (*src.lookoutequipment.plot.TimeSeriesVisualization method*), 26
__init__() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 20
__init__() (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector method*), 23

A

add_labels() (*src.lookoutequipment.plot.TimeSeriesVisualization method*), 27
add_predictions() (*src.lookoutequipment.plot.TimeSeriesVisualization method*), 27
add_rolling_average() (*src.lookoutequipment.plot.TimeSeriesVisualization method*), 27
add_signal() (*src.lookoutequipment.plot.TimeSeriesVisualization method*), 27
add_train_test_split() (*src.lookoutequipment.plot.TimeSeriesVisualization method*), 27

B

build_inspection_report() (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector method*), 23

C

compute_histograms() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 18
create() (*src.lookoutequipment.dataset.LookoutEquipmentDataset method*), 13
create() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 20
create_data_schema() (*in module src.lookoutequipment.schema*), 10
create_data_schema_from_dir() (*in module src.lookoutequipment.schema*), 9
create_data_schema_from_s3_path() (*in module src.lookoutequipment.schema*), 9
create_model_request (*src.lookoutequipment.model.LookoutEquipmentModel attribute*), 15
create_request (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler attribute*), 20
current_time (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 23

D

dataset_name (*src.lookoutequipment.model.LookoutEquipmentModel attribute*), 14
delay_offset (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler attribute*), 22
delete() (*src.lookoutequipment.dataset.LookoutEquipmentDataset method*), 13
delete() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 15
delete() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 20
delimiter (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 22
df_list (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis attribute*), 17

E

end_time (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 23
execution_summaries (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler attribute*), 20
export_to_html() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 24

F

frequency (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 22

G

generate_replay_data() (*in module src.lookoutequipment.dataset*), 12
get_component_field_map() (*src.lookoutequipment.dataset.LookoutEquipmentDataset method*), 13
get_labels() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 18
get_next_time_range() (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector method*), 24
get_predictions() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 18
get_predictions() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 21
get_ranked_list() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 18
get_status() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 21

I

ingest_data() (*src.lookoutequipment.dataset.LookoutEquipmentDataset method*), 13
input_timezone_offset (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 22

L

labelled_ranges (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis attribute*), 17
list_datasets() (*in module src.lookoutequipment.dataset*), 10
list_inference_executions() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 21
list_models() (*in module src.lookoutequipment.model*), 14
list_models() (*src.lookoutequipment.dataset.LookoutEquipmentDataset method*), 14
load_dataset() (*in module src.lookoutequipment.dataset*), 10
LookoutEquipmentAnalysis (*class in src.lookoutequipment.evaluation*), 17
LookoutEquipmentDataset (*class in src.lookoutequipment.dataset*), 12
LookoutEquipmentModel (*class in src.lookoutequipment.model*), 14
LookoutEquipmentScheduler (*class in src.lookoutequipment.scheduler*), 20
LookoutEquipmentSchedulerInspector (*class in src.lookoutequipment.scheduler*), 22

M

model_name (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis attribute*), 17
model_name (*src.lookoutequipment.model.LookoutEquipmentModel attribute*), 14
model_name (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler attribute*), 20
module
src.lookoutequipment.dataset, 10
src.lookoutequipment.evaluation, 17
src.lookoutequipment.model, 14
src.lookoutequipment.plot, 24
src.lookoutequipment.scheduler, 19
src.lookoutequipment.schema, 9

N

next_timestamp (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler attribute*), 23
next_wakeup_time (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler attribute*), 23

P

plot() (*src.lookoutequipment.plot.TimeSeriesVisualization method*), 27
plot_event_barh() (*in module src.lookoutequipment.plot*), 25
plot_histogram_comparison() (*in module src.lookoutequipment.plot*), 24
plot_histograms() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 18
plot_histograms() (*src.lookoutequipment.plot.TimeSeriesVisualization method*), 28
plot_histograms_v2() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 19
plot_range() (*in module src.lookoutequipment.plot*), 25
plot_signals() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 19
poll_data_ingestion() (*src.lookoutequipment.dataset.LookoutEquipmentDataset method*), 14
poll_model_training() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 15
predicted_ranges (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis attribute*), 17
prepare_inference_data() (*in module src.lookoutequipment.dataset*), 11

S

s3_input_location (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler attribute*), 22
scheduler_name (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler attribute*), 20

scheduler_name (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 22
set_label_data() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 15
set_off_condition() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 16
set_off_conditions() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 16
set_parameters() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 21
set_subset_schema() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 16
set_target_sampling_rate() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 16
set_time_periods() (*src.lookoutequipment.evaluation.LookoutEquipmentAnalysis method*), 19
set_time_periods() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 16
src.lookoutequipment.dataset
    module, 10
src.lookoutequipment.evaluation
    module, 17
src.lookoutequipment.model
    module, 14
src.lookoutequipment.plot
    module, 24
src.lookoutequipment.scheduler
    module, 19
src.lookoutequipment.schema
    module, 9
start() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 22
start_time (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 23
stop() (*src.lookoutequipment.scheduler.LookoutEquipmentScheduler method*), 22

T

TimeSeriesVisualization (*class in src.lookoutequipment.plot*), 25
timestamp_format (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 22
timezone (*src.lookoutequipment.scheduler.LookoutEquipmentSchedulerInspector attribute*), 22
train() (*src.lookoutequipment.model.LookoutEquipmentModel method*), 16

U

upload_dataset() (*in module src.lookoutequipment.dataset*), 11