python_moztelemetry Documentation

Release 0.3.9.13

Mozilla Firefox Data Platform

Jul 12, 2018

Contents

A simple library to fetch and analyze data collected by the Mozilla Telemetry service. Objects collected by Telemetry are called `pings`. A ping has a number of properties (aka `dimensions`) and a payload.

A session of Telemetry data analysis/manipulation typically starts with a *Dataset* query that filters the objects by one or more dimensions, and then extracts the items of interest from their payload.

CHAPTER 1

API

1.1 Dataset

class moztelemetry.dataset.**Dataset** (*bucket*, *schema*, *store=None*, *prefix=None*, *clauses=None*, *selection=None*)

Represents a collection of objects on S3.

A Dataset can have zero, one or many filters, which are refined using the *where* method. The result of refining a Dataset is a Dataset itself, so it's possible to chain multiple *where* clauses together.

The actual data retrieval is triggered by the *records* method, which returns a Spark RDD containing the list of records retrieved. To call *records* a SparkContext object must be provided.

Usage example:

```
bucket = 'test-bucket'
schema = ['submissionDate', 'docType', 'platform']

records = Dataset(bucket, schema) \
    .select(
        'clientId',
        os_name='environment.system.os.name',
        first_paint='payload.simpleMeasurements.firstPaint',
        // Take the first 2 stacks for each thread hang.
        stack_list='payload.threadHangStats[].hangs[].stack[0:2]'
    ).where(
        docType='main',
        appUpdateChannel='nightly',
        submissionDate=lambda x: x.startswith('201607'),
    ).records(sc)
```

For convenience Dataset objects can be created using the factory method *from_source*, that takes a source name (e.g. 'telemetry') and returns a new Dataset instance. The instance created will be aware of the list of dimensions, available on its *schema* attribute for inspection.

static from_source()

Create a Dataset configured for the given source_name

This is particularly convenient when the user doesn't know the list of dimensions or the bucket name, but only the source name.

Usage example:

```
records = Dataset.from_source('telemetry').where(
    docType='main',
    submissionDate='20160701',
    appUpdateChannel='nightly'
)
```

records(*sc*, *limit=None*, *sample=1*, *seed=42*, *decode=None*, *summaries=None*)

Retrieve the elements of a Dataset

Parameters

- **sc** – a SparkContext object
- **limit** – maximum number of objects to retrieve
- **decode** – an optional transformation to apply to the objects retrieved
- **sample** – percentage of results to return. Useful to return a sample of the dataset. This parameter is ignored when *limit* is set.
- **seed** – initialize internal state of the random number generator (42 by default). This is used to make the dataset sampling reproducible. It can be set to None to obtain different samples.
- **summaries** – an iterable containing a summary for each item in the dataset. If None, it will computed calling the summaries dataset.

Returns a Spark rdd containing the elements retrieved

select(**properties*, ***aliased_properties*)

Specify which properties of the dataset must be returned

Property extraction is based on JMESPath expressions. This method returns a new Dataset narrowed down by the given selection.

Parameters

- **properties** – JMESPath to use for the property extraction. The JMESPath string will be used as a key in the output dictionary.
- **aliased_properties** – Same as properties, but the output dictionary will contain the parameter name instead of the JMESPath string.

summaries(*sc*, *limit=None*)

Summary of the files contained in the current dataset

Every item in the summary is a dict containing a key name and the corresponding size of the key item in bytes, e.g.:: {'key': 'full/path/to/my/key', 'size': 200}

Parameters **limit** – Max number of objects to retrieve

Returns An iterable of summaries

where(***kwargs*)

Return a new Dataset refined using the given condition

Parameters **kwargs** – a map of *dimension* => *condition* to filter the elements of the dataset. *condition* can either be an exact value or a callable returning a boolean value. If *condition* is a value, it is converted to a string, then sanitized.

1.2 Deprecated ping methods

Before the Dataset API was available, a number of custom methods were written for selecting a set of telemetry pings and extracting data from them. These methods are somewhat convoluted and difficult to understand, and are not recommended for new code.

moztelemetry.spark.**get_pings**(**args*, ***kwargs*)

Returns a RDD of Telemetry submissions for a given filtering criteria.

Parameters

- **sc** – an instance of SparkContext
- **app** – an application name, e.g.: "Firefox"
- **channel** – a channel name, e.g.: "nightly"
- **version** – the application version, e.g.: "40.0a1"
- **build_id** – a build_id or a range of build_ids, e.g.: "20150601000000" or ("20150601000000", "20150610999999")
- **submission_date** – a submission date or a range of submission dates, e.g: "20150601" or ("20150601", "20150610")
- **source_name** – source name, set to "telemetry" by default
- **source_version** – source version, set to "4" by default
- **doc_type** – ping type, set to "saved_session" by default
- **schema** – (deprecated) version of the schema to use
- **fraction** – the fraction of pings to return, set to 1.0 by default

moztelemetry.spark.**get_pings_properties**(**args*, ***kwargs*)

Returns a RDD of a subset of properties of pings. Child histograms are automatically merged with the parent histogram.

If one of the paths points to a keyedHistogram name without supplying the actual key, returns a dict of all available subhistograms for that property.

Parameters

- **with_processes** – should separate parent and child histograms be included as well?
- **paths** – paths to properties in the payload, with levels separated by "/". These can be supplied either as a list, eg. ["application/channel", "payload/info/subsessionStartDate"], or as the values of a dict keyed by custom identifiers, eg. {"channel": "application/channel", "ssd": "payload/info/subsessionStartDate"}.
- **histograms_url** – see histogram.Histogram constructor
- **additional_histograms** – see histogram.Histogram constructor

The returned RDD contains a dict for each ping with the required properties as values, keyed by the original paths (if 'paths' is a list) or the custom identifier keys (if 'paths' is a dict).

`moztelemetry.spark.get_one_ping_per_client(*args, **kwargs)`

Returns a single ping for each client in the RDD.

THIS METHOD IS NOT RECOMMENDED: The ping to be returned is essentially selected at random. It is also expensive as it requires data to be shuffled around. It should be run only after extracting a subset with get_pings_properties.

1.3 Using Spark RDDs

Both `Dataset` and `get_pings` return the data as a Spark RDD. Users can then use the RDD api to further shape or transform the dataset.

1.4 Experimental APIs

`moztelemetry.zeppelin.show(fig, width=600)`

Renders a Matplotlib figure in Zeppelin.

Parameters

- **`fig`** – a Matplotlib figure
- **`width`** – the width in pixel of the rendered figure, defaults to 600

Usage example:

```
import matplotlib.pyplot as plt
from moztelemetry.zeppelin import show

fig = plt.figure()
plt.plot([1, 2, 3])
show(fig)
```

Members

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index