
datamonster-api

Feb 03, 2020

Contents:

1	Quickstart	3
2	Interface	5
3	Objects	9
4	Examples	13
4.1	Get Data Raw	13
4.2	Get Dimensions for Datasource	15
	Python Module Index	17
	Index	19

Release v0.5.2. ([QuickStart](#))

This library eases the use of the Datamonitor REST API via Python

CHAPTER 1

Quickstart

Installing the Datamonster library:

```
pip install datamonster_api
```

Working with companies:

```
from datamonster_api import DataMonster
dm = DataMonster(<key_id>, <secret_key>)

# Prints all companies whose name or ticker matches 'hd'
print(list(dm.get_companies(query='hd')) )

# Creates a company object for apple
apple = dm.get_company_by_ticker('aapl')

# prints the first 5 quarter end dates
print(apple.quarters[:5])

# prints the first 5 data sources that cover apple
print(list(apple.datasources)[:5])
```

Working with data sources:

```
# Prints all data sources whose name or provider matches 'fake'
print(list(dm.get_datasources(query='fake')) )

# Prints all data sources whose name or provider matches 'fake'
# AND also cover apple
print(list(dm.get_datasources(query='fake', company=apple)))

# Prints first 5 companies covered by `Fake Data Source`
datasource = list(
    dm.get_datasources(query='Fake Data Source')
```

(continues on next page)

(continued from previous page)

```
    ) [0]  
  
print(list(datasource.companies)[:5])
```

Getting data:

```
import datetime  
from datamonster_api import Aggregation  
  
# Gets a datasource object  
apple = dm.get_company_by_ticker('aapl')  
datasource = next(apple.datasources)  
  
# Gets all data for the data source filtering on apple  
datasource.get_data(apple)  
  
agg = Aggregation(period='fiscalQuarter', company=apple)  
  
# Gets all data for the given data source filtered by apple,  
# aggregated by apple's fiscal quarter, and starting on  
# January 1, 2017 (inclusive)  
datasource.get_data(  
    apple,  
    agg,  
    start_date=datetime.date(2017, 1, 1)  
)
```

This part of the documentation covers all the interfaces of the datamonster-api.

class `datamonster_api.DataMonster` (*key_id, secret, server=None, verify=True*)

DataMonster object. Main entry point to the library

Parameters

- **key_id** – (str) a user’s public key
- **secret** – (str) a user’s secret key
- **server** – (optional, str) default to `dm.adaptivemgmt.com`
- **verify** – (optional, bool) whether to verify the server’s TLS certificate

get_companies (*query=None, datasource=None*)

Get available companies

Parameters

- **query** – Optional query that will restrict companies by ticker or name
- **datasource** – Optional `Datasource` object that restricts companies to those covered by the given data source

Returns Iterator of `Company` objects

get_company_by_id (*company_id*)

Get a single company by id

Parameters **company_id** – (str or int) unique internal identifier for the desired company. Can take str form e.g. ‘718’, or int form, e.g. 707. In order to find the id of a frequently used company, find the company by ticker and call `.pk` on the resulting `Company` object

Returns Single `Company` object if any company matches the id

Raises `DataMonsterError` if no company matches id

get_company_by_ticker (*ticker*)

Get a single company by ticker

Parameters **ticker** – Ticker to search for

Returns Single `Company` object if any companies exactly match the ticker (case insensitive)

Raises `DataMonsterError` if no companies match ticker

get_company_details (*company_id*)

Get details for the given company

Parameters **company_id** – (str or int) unique internal identifier for company. See the method [get_company_by_id](#) for more info on `company_id`.

Returns (dict) details (metadata) for this company, providing basic information.

get_data (*datasource, company, aggregation=None, start_date=None, end_date=None*)

Get data for data source

Parameters

- **datasource** – `Datasource` object to get the data for
- **company** – `Company` object to filter the data source on
- **aggregation** – Optional `Aggregation` object to specify the aggregation of the data
- **start_date** – Optional filter for the start date of the data
- **end_date** – Optional filter for the end date of the data

See [here](#) for example usage.

Returns `pandas.DataFrame`

get_data_group_by_id (*id*)

Give a data group pk (primary key), return the corresponding `DataGroup` object. To find the pk for a data group, first find it using the iterator returned by `get_data_groups()`, then call `.id` on the `DataGroup` object.

Parameters **id** – (int)

Returns Single `DataGroup` object with the given id

Raises `DataMonsterError` if no data group matches the given id

get_data_group_details (*id*)

Given a data group id, return the corresponding `DataGroup` object

Parameters **id** – (int)

Returns Single `DataGroup` object with the given id

Raises `DataMonsterError` if no data group matches the given id

get_data_groups (*query=None*)

Get available data groups

Parameters **query** – (str) Optional query that will restrict data groups by name or data source name

Returns Iterator of `DataGroup` objects.

get_data_raw (*datasource, filters=None, aggregation=None*)

Get raw data for all companies available in the data source.

Parameters

- **datasource** – `Datasource` object to get the data for
- **aggregation** – `Aggregation` object to specify requested aggregation

- **filters** – dictionary of requested filters

Returns (schema, pandas.DataFrame)

See [here](#) for example usage.

get_datasource_by_id (*datasource_id*)

Given a data source UUID (universal unique identifier), return the corresponding *Datasource* object. To find the UUID for a data source, first find it by name, then call `.id` on the resulting *Datasource* object.

Parameters **datasource_id** – (str)

Returns Single *Datasource* object with the given id

Raises *DataMonsterError* if no data source matches the given id

get_datasource_by_name (*name*)

Given a name, try to find a data source of that name

Parameters **name** – (str)

Returns Single *Datasource* object with the given name

Raises *DataMonsterError* if no data source matches the given name

get_datasource_details (*datasource_id*)

Get details (metadata) for the data source corresponding to the given UUID

Parameters **datasource_id** – (str) See the method [get_datasource_by_id](#) for more info on `datasource_id`

Returns (dict) details (metadata) for this data source, providing basic information.

get_datasources (*query=None, company=None*)

Get available datasources

Parameters

- **query** – (str) Optional query that will restrict data sources by name or provider name
- **company** – Optional *Company* object that restricts data sources to those that cover the given company

Returns Iterator of *Datasource* objects

get_dimensions_for_datasource (*datasource, filters=None, add_company_info_from_pks=False*)

Get dimensions (“splits”) for the data source from the DataMonster REST endpoint `/datasource/<uuid>/dimensions?filters=...` where the `filters` string is optional.

Parameters

- **datasource** – *Datasource* object
- **filters** – (dict): a dict of key/value pairs to filter dimensions by
- **add_company_info_from_pks** – (bool): Determines whether return value will include tickers for the returned companies. If `False`, only `section_pks` will be returned.

See [here](#) for example usage.

Returns a *DimensionSet* object - an iterable through a collection of dimension dicts, filtered as requested. See [this documentation](#) for more info on *DimensionSet* objects.

Raises `DataMonsterError` if `filters` is not a dict or is not JSON-serializable. Re-raises `DataMonsterError` if `self.client.get()` raises that.

`get_raw_data(*args, **kwargs)`

This function is deprecated. Please use the `get_data_raw` function instead

class `datamonster_api.Company` (*_id, ticker, name, uri, dm*)

Representation of a company in DataMonster

Parameters

- **_id** – (str) unique internal identifier for the company
- **ticker** – (str) ticker of the company
- **name** – (str) name of the company
- **uri** – (str) DataMonster resource identifier associated with the company
- **dm** – DataMonster object

property ticker **Returns:** (str) company ticker, None if company is private

property name **Returns:** (str) company name, including the associated vendor

property quarters

Returns: (list) list of company quarter dates, including 4 projected dates. Empty if company is private

datasources

Returns (iter) iterable of `Datasource` objects associated with this company to which the user has access, memoized

get_details ()

Get details (metadata) for this company, providing basic information as stored in DataMonster

Returns (dict)

pk

Returns (int) the unique internal identifier for the company (corresponds to `_id`)

class `datamonster_api.Datasource` (*_id, name, category, uri, dm*)

Representation of a data source in DataMonster

Parameters

- **_id** – (str) unique internal identifier for the data source
- **name** – (dict) name of the data source, including the vendor for the data
- **category** – (list) associated categories
- **uri** – (str) DataMonster resource identifier associated with the data source
- **dm** – DataMonster object

property name Returns (str) name of data source, including vendor

property category Returns (str) category associated with the data source, e.g., *Web Scrape Data* or *Uploaded Data*

companies

Returns (iter) iterable of `Company` objects associated with this data source, memoized

get_data (*company*, *aggregation=None*, *start_date=None*, *end_date=None*)
Get data for this data source.

Parameters

- **company** – `Company` object to filter the data source on
- **aggregation** – Optional `Aggregation` object to specify the aggregation of the data
- **start_date** – Optional string to act as a filter for the start date of the data; accepted formats include: YYYY-MM-DD, MM/DD/YYYY, or pandas or regular `datetime` object
- **end_date** – Optional string to act as a filter for the end date of the data; accepted formats include: YYYY-MM-DD or MM/DD/YYYY, or pandas or regular `datetime` object

Returns `pandas.DataFrame`

get_details ()
Get details (metadata) for this data source, providing basic information as stored in DataMonster

Returns (dict)

get_dimensions (*company=None*, *add_company_info_from_pks=True*, ***kwargs*)

Return the dimensions for this data source, restricted to the given company or companies and filtered by any `kwargs` items. Not memoized.

Parameters

- **company** – a `Company` object, a list or tuple of `Company` objects, or `None`. If not `None` the return value will only include rows corresponding to the given companies.
- **add_company_info_from_pks** – Determines whether return value will include tickers for the returned companies. If `False`, only `section_pks` will be returned.
- **kwargs** – Additional items to filter by, e.g. `category='Banana Republic'`

Returns

a `DimensionSet` object - an iterable through a collection of dimension dicts, filtered as requested. See [this documentation](#) for more info.

See [here](#) for example usage of a similar function.

Raises can raise `DataMonsterError` if company is not of an expected type, or if some kwarg item is not JSON-serializable.

class `datamonster_api.DimensionSet` (*url, dm, add_company_info_from_pks*)

An iterable through a collection of dimensions dictionaries.

Each dimension dictionary has 4 keys: `max_date`, `min_date`, `row_count`, and `split_combination`. The first two have values that are dates as strings in ISO format; `split_combination` points to a dict containing data from all other columns; `row_count` points to an int specifying how many rows match the dates and all splits in `split_combination`

has_extra_company_info

Returns (bool) The value passed as `add_company_info_from_pks` to the constructor, coerced to *bool*.

max_date

Returns (str) max of the `max_date` of the dimension dicts

min_date

Return type (str) min of the `min_date` of the dimension dicts

row_count

Returns (int) number of rows matching the filters for this `DimensionSet`

class `datamonster_api.Aggregation` (*period, company*)

A representation of an aggregation type within `DataMonster`

4.1 Get Data Raw

Initialize a `DataMonster` object:

```
dm = DataMonster(<key_id>, <secret_key>)
```

Initialize a `Datasource` object (we will use a fake small data source from the provider XYZ for the purposes of this example):

```
ds = dm.get_datasource_by_name(  
    'XYZ Data Source'  
)
```

Get raw data from the data source, producing a schema and pandas dataframe:

```
schema, df = dm.get_data_raw(ds)
```

The schema will contain metadata for the data source, with keys and values showing the roles different columns play in the data. In the case of the above data source:

```
>>> schema  
{  
    'lower_date': ['period_start'],  
    'upper_date': ['period_end'],  
    'section_pk': ['section_pk'],  
    'value': ['panel_sales'],  
    'split': ['category']  
}
```

This result indicates that the `period_start` column represents the lower date for each data point, and so on.

Next, looking at the dataframe we see:

```
>>> df.head(2)
```

category	panel_sales	period_end	period_start	section_pk
Not specified	-0.1139	2017-01-01	2016-10-02	617
Not Specified	-0.0523	2018-07-02	2018-04-02	742
Category1	-0.2233	2018-07-02	2018-04-02	742
Category1	-0.4132	2019-03-31	2019-01-01	205

Note that the `section_pk` column, which represents which company each data point refers to, is currently in the form of an internal DataMonster identifier and is not particularly useful for external use. To convert to a more usable form, try:

```
comps = ds.companies
section_map = {}
for comp in comps:
    section_map[comp.pk] = {"name": comp.name,
                           "ticker": comp.ticker}

def map_pk_to_ticker_and_name(section_map, df):
    ticker_dict = {
        pk: v["ticker"] for pk, v in section_map.items()
    }

    name_dict = {
        pk: v["name"] for pk, v in section_map.items()
    }

    df["ticker"] = df["section_pk"].map(ticker_dict)
    df["comp_name"] = df["section_pk"].map(name_dict)

    df = df.drop(["section_pk"], axis=1)

    return df
```

We can now use `map_pk_to_ticker_and_name` to produce a more human-readable dataframe. For example:

```
>>> map_pk_to_ticker_and_name(section_map, df).head(2)
```

category	panel_sales	period_end	period_start	ticker	comp_name
Not specified	-0.1139	2017-01-01	2016-10-02	PRTY	PARTY CITY
Not Specified	-0.0523	2018-07-02	2018-04-02	RUTH	RUTH'S HOSPITALITY GROUP
Category1	-0.2233	2018-07-02	2018-04-02	RUTH	RUTH'S HOSPITALITY GROUP
Category1	-0.4132	2019-03-31	2019-01-01	HD	HOME DEPOT

4.1.1 Filtering to Specific Dimensions

The raw data endpoint supports filtering to specific values for dimensions by applying key value pairs as a dictionary, where the key is the dimension name and the value is a list of possibilities for that dimension. Using the example above, we could do this in a variety of ways.

Filtering to specific companies (in this case, Party City and Home Depot):

```
>>> filters = {'section_pk': [617, 205]}
>>> schema, df = dm.get_data_raw(ds, filters=filters)
```

category	panel_sales	period_end	period_start	section_pk
Not specified	-0.1139	2017-01-01	2016-10-02	617
Category1	-0.4132	2019-03-31	2019-01-01	205

Filtering to specific dimension values (in this case, "Category1"):

```
>>> filters = {'category': ['Category1']}
>>> schema, df = dm.get_data_raw(ds, filters=filters)
```

category	panel_sales	period_end	period_start	section_pk
Category1	-0.2233	2018-07-02	2018-04-02	742
Category1	-0.4132	2019-03-31	2019-01-01	205

Combining filters across dimensions (in this case, "Category1" for Ruth's Hospitality Group):

```
>>> filters = {'section_pk': [742], 'category': ['Category1']}
>>> schema, df = dm.get_data_raw(ds, filters=filters)
```

category	panel_sales	period_end	period_start	section_pk
Category1	-0.2233	2018-07-02	2018-04-02	742

4.1.2 Aggregating Results on Different Cadences

The raw data endpoint can also take an optional `Aggregation` object to request data with a time-based aggregation applied. For example:

```
from datamonster_api import DataMonster, Aggregation

dm = DataMonster(<key_id>, <secret_key>)

# Get Company for Home Depot
hd = dm.get_company_by_ticker('hd')

# Get our Data Source
ds = dm.get_datasource_by_name('XYZ Data Source')

# Filter to Home Depot data and aggregate by Home Depot's fiscal quarters
filters = {'section_pk': [hd.pk]}
agg = Aggregation(period='fiscalQuarter', company=hd)
dm.get_data_raw(ds, filters=filters, aggregation=agg)
```

4.2 Get Dimensions for Datasource

Assuming `dm` is a `DataMonster` object, and given this fake data source and company:

```
datasource = next(
    dm.get_datasources(query="Fake Data Source")
)
the_gap = dm.get_company_by_ticker("GPS")
```

this call to `get_dimensions_for_datasource`:

```
dimset = dm.get_dimensions_for_datasource(
    datasource,
    filters={
        "section_pk": the_gap.pk,
        "category": "Banana Republic",
    },
)
```

returns an iterable, `dimset`, to a collection with just one dimensions dict. Assuming from `pprint` import `pprint`, the following loop:

```
for dim in dimset:
    pprint(dim)
```

prettyprints the single dimension dict:

```
{
    "max_date": "2019-06-21",
    "min_date": "2014-01-01",
    "row_count": 1998,
    "split_combination": {
        "category": "Banana Republic",
        "country": "US",
        "section_pk": 707,
    },
}
```

d

`datamonster_api`, 4

A

Aggregation (*class in datamonster_api*), 11

C

companies (*datamonster_api.Datasource attribute*), 10

Company (*class in datamonster_api*), 9

D

DataMonster (*class in datamonster_api*), 5

datamonster_api (*module*), 4

Datasource (*class in datamonster_api*), 9

datasources (*datamonster_api.Company attribute*), 9

DimensionSet (*class in datamonster_api*), 11

G

get_companies() (*datamonster_api.DataMonster method*), 5

get_company_by_id() (*datamonster_api.DataMonster method*), 5

get_company_by_ticker() (*datamonster_api.DataMonster method*), 5

get_company_details() (*datamonster_api.DataMonster method*), 6

get_data() (*datamonster_api.DataMonster method*), 6

get_data() (*datamonster_api.Datasource method*), 10

get_data_group_by_id() (*datamonster_api.DataMonster method*), 6

get_data_group_details() (*datamonster_api.DataMonster method*), 6

get_data_groups() (*datamonster_api.DataMonster method*), 6

get_data_raw() (*datamonster_api.DataMonster method*), 6

get_datasource_by_id() (*datamonster_api.DataMonster method*), 7

get_datasource_by_name() (*datamonster_api.DataMonster method*), 7

get_datasource_details() (*datamonster_api.DataMonster method*), 7

get_datasources() (*datamonster_api.DataMonster method*), 7

get_details() (*datamonster_api.Company method*), 9

get_details() (*datamonster_api.Datasource method*), 10

get_dimensions() (*datamonster_api.Datasource method*), 10

get_dimensions_for_datasource() (*datamonster_api.DataMonster method*), 7

get_raw_data() (*datamonster_api.DataMonster method*), 8

H

has_extra_company_info (*datamonster_api.DimensionSet attribute*), 11

M

max_date (*datamonster_api.DimensionSet attribute*), 11

min_date (*datamonster_api.DimensionSet attribute*), 11

P

pk (*datamonster_api.Company attribute*), 9

R

row_count (*datamonster_api.DimensionSet attribute*), 11