Tiingo Python Documentation

Release 0.14.0

Cameron Yick

Apr 02, 2022

Contents

1 Tiingo Python **3**
- 1.1 Usage . . . 3
- 1.2 Cryptocurrency . . . 5
- 1.3 Websockets Support . . . 6
- 1.4 Further Docs . . . 6
- 1.5 Features . . . 7
- 1.6 Roadmap: . . . 7
- 1.7 Related Projects: . . . 7
- 1.8 Credits . . . 7

2 Installation **9**
- 2.1 Stable release . . . 9
- 2.2 From sources . . . 9

3 Usage **11**
- 3.1 Further Docs . . . 12

4 Contributing **13**
- 4.1 Types of Contributions . . . 13
- 4.2 Get Started! . . . 14
- 4.3 Pull Request Guidelines . . . 15
- 4.4 Release Procedure . . . 15
- 4.5 Tips . . . 15

5 Credits **17**
- 5.1 Development Lead . . . 17
- 5.2 Contributors . . . 17

6 History **19**
- 6.1 0.15.0 (2021-XX - Unreleased) . . . 19
- 6.2 0.14.0 (2021-03-06) . . . 19
- 6.3 0.13.0 (2020-12-12) . . . 19
- 6.4 0.12.0 (2019-10-20) . . . 20
- 6.5 0.11.0 (2019-09-01) . . . 20
- 6.6 0.10.x (2019-05-11) . . . 20
- 6.7 0.9.x (2019-01-30) . . . 20
- 6.8 0.8.0 (2018-07-06) . . . 20

6.9 0.7.0 (2018-06-14) 20
6.10 0.6.0 (2018-04-30) 20
6.11 0.5.0 (2018-03-11) 21
6.12 0.4.0 (2017-10-22) 21
6.13 0.3.0 (2017-09-17) 21
6.14 0.2.0 (2017-09-01) 21
6.15 0.1.0 (2017-08-24) 21

7 Indices and tables **23**

Contents:

CHAPTER 1

Tiingo Python

Tiingo is a financial data platform that makes high quality financial tools available to all. Tiingo has a REST and Real-Time Data API, which this library helps you to access. Presently, the API includes support for the following endpoints:

- Stock Market Ticker Closing Prices + Metadata. Data includes full distribution details and is validated using a proprietary EOD Price Engine.
- Curated news from top financial news sources + blogs. Stories are tagged with topic tags and relevant stock tickers by Tiingo's algorithms.

1.1 Usage

If you'd like to try this library before installing, click below to open a folder of online runnable examples.

First, install the library from PyPi:

```
pip install tiingo
```

If you prefer to receive your results in `pandas DataFrame` or `Series` format, and you do not already have pandas installed, install it as an optional dependency:

```
pip install tiingo[pandas]
```

Next, initialize your client. It is recommended to use an environment variable to initialize your client for convenience.

```
from tiingo import TiingoClient
# Set TIINGO_API_KEY in your environment variables in your .bash_profile, OR
# pass a dictionary with 'api_key' as a key into the TiingoClient.

client = TiingoClient()
```

Alternately, you may use a dictionary to customize/authorize your client.

```
config = {}

# To reuse the same HTTP Session across API calls (and have better performance),␣
↪include a session key.
config['session'] = True

# If you don't have your API key as an environment variable,
# pass it in via a configuration dictionary.
config['api_key'] = "MY_SECRET_API_KEY"

# Initialize
client = TiingoClient(config)
```

Now you can use `TiingoClient` to make your API calls. (Other parameters are available for each endpoint beyond what is used in the below examples, inspect the docstring for each function for details.).

```
# Get Ticker
ticker_metadata = client.get_ticker_metadata("GOOGL")

# Get latest prices, based on 3+ sources as JSON, sampled weekly
ticker_price = client.get_ticker_price("GOOGL", frequency="weekly")

# Get historical GOOGL prices from August 2017 as JSON, sampled daily
historical_prices = client.get_ticker_price("GOOGL",
                                            fmt='json',
                                            startDate='2017-08-01',
                                            endDate='2017-08-31',
                                            frequency='daily')

# Check what tickers are available, as well as metadata about each ticker
# including supported currency, exchange, and available start/end dates.
tickers = client.list_stock_tickers()

# Get news articles about given tickers or search terms from given domains
articles = client.get_news(tickers=['GOOGL', 'AAPL'],
                           tags=['Laptops'],
                           sources=['washingtonpost.com'],
                           startDate='2017-01-01',
                           endDate='2017-08-31')

# Get definitions for fields available in the fundamentals-api, ticker is
# optional
definitions = client.get_fundamentals_definitions('GOOGL')

# Get fundamentals which require daily-updated (like marketCap). A start-
# and end-date can be passed. If omited, will get all available data.
fundamentals_daily = client.get_fundamentals_daily('GOOGL',
                                        startDate='2020-01-01',
                                        endDate='2020-12-31')
```

(continues on next page)

(continued from previous page)

```

# Get fundamentals based on quarterly statements. Accepts time-range like
# daily-fundamentals. asReported can be set to get the data exactly like
# it was reported to SEC. Set to False if you want to get data containing
# corrections
fundamentals_stmnts = client.get_fundamentals_statements('GOOGL',
                                                         startDate='2020-01-01',
                                                         endDate='2020-12-31',
                                                         asReported=True)
```

To receive results in `pandas` format, use the `get_dataframe()` method:

```
#Get a pd.DataFrame of the price history of a single symbol (default is daily):
ticker_history = client.get_dataframe("GOOGL")

#The method returns all of the available information on a symbol, such as open, high,␣
↪low, close,
#adjusted close, etc.  This page in the tiingo api documentation lists the available␣
↪information on each
#symbol: https://api.tiingo.com/docs/tiingo/daily#priceData.

#Frequencies and start and end dates can be specified similarly to the json method␣
↪above.

#Get a pd.Series of only one column of the available response data by specifying one␣
↪of the valid the
#'metric_name' parameters:
ticker_history = client.get_dataframe("GOOGL", metric_name='adjClose')

#Get a pd.DataFrame for a list of symbols for a specified metric_name (default is␣
↪adjClose if no
#metric_name is specified):
ticker_history = client.get_dataframe(['GOOGL', 'AAPL'],
                                      frequency='weekly',
                                      metric_name='volume',
                                      startDate='2017-01-01',
                                      endDate='2018-05-31')
```

You can specify any of the end of day frequencies (daily, weekly, monthly, and annually) or any intraday frequency for both the `get_ticker_price` and `get_dataframe` methods. Weekly frequencies resample to the end of day on Friday, monthly frequencies resample to the last day of the month, and annually frequencies resample to the end of day on 12-31 of each year. The intraday frequencies are specified using an integer followed by "Min" or "Hour", for example "30Min" or "1Hour".

1.2 Cryptocurrency

```
# You can obtain cryptocurrency metadata using the following method.
# NOTE: Crypto symbol MUST be encapsulated in brackets as a Python list!

client.get_crypto_metadata(['BTCUSD'], fmt='json')

#You can obtain top-of-book cryptocurrency quotes from the ``get_crypto_top_of_
↪book()`` method.
# NOTE: Crypto symbol MUST be encapsulated in brackets as a Python list!
```

(continues on next page)

(continued from previous page)

```
crypto_price = client.get_crypto_top_of_book(['BTCUSD'])``

# You can obtain historical Cryptocurrency price quotes from the get_crypto_price_
↪history() method.
# NOTE: Crypto symbol MUST be encapsulated in brackets as a Python list!

client.get_crypto_price_history(tickers = ['BTCUSD'], startDate='2020-12-2',
                                endDate='2020-12-3', resampleFreq='1Hour')
```

1.3 Websockets Support

```
from tiingo import TiingoWebsocketClient

def cb_fn(msg):

    # Example response
    # msg = {
    #   "service":"iex" # An identifier telling you this is IEX data.
    #   The value returned by this will correspond to the endpoint argument.
    #
    #   # Will always return "A" meaning new price quotes. There are also H type␣
↪Heartbeat msgs used to keep the connection alive
    #   "messageType":"A" # A value telling you what kind of data packet this is from␣
↪our IEX feed.
    #
    #   # see https://api.tiingo.com/documentation/websockets/iex > Response for more␣
↪info
    #   "data":[] # an array containing trade information and a timestamp
    #
    # }

    print(msg)

subscribe = {
        'eventName':'subscribe',
        'authorization':'API_KEY_GOES_HERE',
        #see https://api.tiingo.com/documentation/websockets/iex > Request for more␣
↪info
        'eventData': {
            'thresholdLevel':5
      }
}

# any logic should be implemented in the callback function (cb_fn)
TiingoWebsocketClient(subscribe,endpoint="iex",on_msg_cb=cb_fn)
```

1.4 Further Docs

- Official Tiingo Documentation: https://api.tiingo.com
- *tiingo-python* Documentation: https://tiingo-python.readthedocs.io.

1.5 Features

- Easy programmatic access to Tiingo API
- Reuse requests session across API calls for better performance
- On most methods, pass in *fmt="object"* as a keyword to have your responses come back as *NamedTuples*, which should have a lower memory impact than regular Python dictionaries.

1.6 Roadmap:

- Client-side validation of tickers
- Data validation of returned responses
- Case insensitivity for ticker names
- More documentation / code examples

Feel free to file a PR that implements any of the above items.

1.7 Related Projects:

- Riingo : Client for Tiingo in the R Programming Language

1.8 Credits

- Many thanks to Rishi Singh for creating Tiingo.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Tiingo Python, run this command in your terminal:

```
$ pip install tiingo
```

This is the preferred method to install Tiingo Python, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Tiingo Python can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/hydrosquall/tiingo-python
```

Or download the tarball:

```
$ curl  -OL https://github.com/hydrosquall-python/tiingo/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Tiingo Python in a project:

```python
import tiingo
```

Next, initialize your client object. It is recommended to use an environment variable to initialize your client object for convenience.

```python
from tiingo import TiingoClient
# Set TIINGO_API_KEY in your environment variables in your .bash_profile, OR
# pass a dictionary with 'api_key' as a key into the TiingoClient.

client = TiingoClient()
```

Alternately, you may use a dictionary to customize/authorize your client.

Now you can use `TiingoClient` to make your API calls. (Other parameters are available for each endpoint beyond what has been written below, see the Tiingo website for full details).

Websocket support:

```
.. code-block:: python
```

from tiingo import TiingoWebsocketClient

def cb_fn(msg):

# Example response # msg = { # “service”:”iex” # An identifier telling you this is IEX data. # The value returned by this will correspond to the endpoint argument. # # # Will always return “A” meaning new price quotes. There are also H type Heartbeat msgs used to keep the connection alive # “messageType”:”A” # A value telling you what kind of data packet this is from our IEX feed. # # # see https://api.tiingo.com/documentation/websockets/iex > Response for more info # “data”:[] # an array containing trade information and a timestamp # # }

print(msg)

subscribe = {

‘eventName’:’subscribe’, ‘authorization’:’API_KEY_GOES_HERE’, #see https://api.tiingo.com/documentation/websockets/iex > Request for more info ‘eventData’: {

‘thresholdLevel’:5

}

} # notice how the object isn’t needed after using it # any logic should be implemented in the callback function TiingoWebsocketClient(subscribe,endpoint=”iex”,on_msg_cb=cb_fn) while True:pass

3.1 Further Docs

- Official Tiingo Documentation: https://api.tiingo.com
- Tiingo-Python Documentation (Under Construction): https://tiingo-python.readthedocs.io.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/hydrosquall/tiingo-python/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

Tiingo Python could always use more documentation, whether as part of the official Tiingo Python docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/hydrosquall/tiingo-python/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *tiingo* for local development.

1. Fork the *tiingo* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/tiingo-python.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv tiingo-python
$ cd tiingo-python/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 tiingo tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.6, and 3.7. These will be checked for you when you open your PR via Github Action checks.

4.4 Release Procedure

See RELEASE.rst .

4.5 Tips

To run a subset of tests:

```
$ py.test tests.test_tiingo
```

To regenerate fixture remove it from tests/fixtures and run tests again:

```
$ rm tests/fixtures/NAME.yaml
$ py.test
```

In order for py.test to run, you will have had to create an environment variable containing a valid tiingo API key so that the test runner can make a valid api call. One way to do that is to:

```
$ export TIINGO_API_KEY='...insert api key here...'
```

However, now this api key will become embedded in the test fixture file that is created per the prior procedure. In order to remove this api key from the new test fixtures, run the following from the top level directory:

```
$ ./tools/api_key_tool.py
```

CHAPTER 5

Credits

5.1 Development Lead

- Cameron Yick <cameron.yick@enigma.com>

5.2 Contributors

- Dmitry Budaev <condemil@gmail.com>
- Bharat Kalluri
- Stephen Clark <steveclarkcode@gmail.com>
- Davis Thames
- Nima Yazdanmehr
- Michael MacCormack
- David Minnen
- Tobias Kaesser <t.kaesser@gmail.com>

CHAPTER 6

History

6.1 0.15.0 (2021-XX - Unreleased)

- Documentation: Add crypto endpoint examples (#621)

6.2 0.14.0 (2021-03-06)

- Feature: Added 3 new methods for fundamentals-endpoint: definitions, daily and statements
- [/news] Fix bug in get_news() when sources list is empty (#566)
- Development: Run tests in Github Actions instead of Travis.org
- Development: This is the last version of tiingo that will support Python 3.5 and below. (#601)

6.3 0.13.0 (2020-12-12)

- Minor: Address Pandas Future Warning for sorting in pd.concat (#392)
- Feature: Add option to request data in csv format in get_dataframe method potentially boosting speed up to 4-5x. (#523)
- Minor: bumped library dependencies, in particular cryptography
- Development: Dropped official support for Python 3.5, replaced with 3.7
- Development: Publish library with Github Actions instead of Travis (#546)

6.4 0.12.0 (2019-10-20)

- Feature: Added 3 new methods for crypo endpoints: top of book prices, historical, and metadata endpoints (@n1rna #340)
- Feature: Permit list_tickers to support multiple asset types at once (@n1rna #346)

6.5 0.11.0 (2019-09-01)

- [/news] Internally rename sources parameter to "source", ensure lists are passed as comma separated values #325. Non-breaking external change.
- [/news] Add new URL parameter for "onlyWithTickers" #327

6.6 0.10.x (2019-05-11)

- Documentation: Added a "Peer Comparison Analysis" Jupyter Notebook under "/examples" (@i3creations #197)
- Minor: Update error message to clarify multiple tickers only work with single metrics
- Updated development dependencies

6.7 0.9.x (2019-01-30)

- Documentation: Added runnable jupyter notebook sample under "/examples"
- Minor: bumped various library dependencies

6.8 0.8.0 (2018-07-06)

- Major: Add IEX Endpoint to retrieve data with intraday frequencies (@dcwtx #125)
- Minor: update documentation for contributing/releasing new versions
- Speed up Travis build process with pip cache

6.9 0.7.0 (2018-06-14)

- Major: Provide functions for returning data as pandas Dataframes or Series (@dcwtx #112)
- Minor documentation edits

6.10 0.6.0 (2018-04-30)

- Fix bug in resample argument name (@dcwtx #82)
- Add tool for removing API Keys from test fixtures (@dcwtx #107)

- Remove official support for Python 3.3

6.11 0.5.0 (2018-03-11)

- Updated examples in docs for getting historical prices
- Add interfaces to obtain mutual fund and ETF tickers (@savagesmc #62)
- Raise explicit error when API key is missing (#44)
- Update development dependencies

6.12 0.4.0 (2017-10-22)

- Make tests run in 1/10th the time with `vcr.py` (@condemil #32)
- Add support for returning python objects instead of dictionaries (@BharatKalluri #33)

6.13 0.3.0 (2017-09-17)

- Add support for listing all tickers + date ranges
- Add support for interacting with the `/news` API
- Improve logging of REST client errors

6.14 0.2.0 (2017-09-01)

- Improve test coverage of tickers endpoint
- Deprecate the Mutual Funds endpoint

6.15 0.1.0 (2017-08-24)

- First release on PyPI.

Tiingo is a financial data platform that makes high quality financial tools available to all. They have a RESTful and Real-Time Data API. Presently, the API includes support for the following endpoints:

- Stock Market Ticker Closing Prices + Metadata. Data includes full distribution details and is validated using a proprietary EOD Price Engine.
- Curated news from top financial news sources + curated blogs. Stories are tagged by Tiingo's algorithms.

If you'd like to try this library without installing anything, click the button below. Otherwise, continue reading.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search