SmartBetas Code Documentation

Release 0.1

Elena Pfefferlé

April 28, 2019

CHAPTER 1

SmartBetas Investing

`__main__` is initiating the application and starting the console.

The module checks if the required library is installed on the host and if the host is connected to the internet. If the tests succeed, it starts the user interface console.

The program is built on a database, the communications between the program and the db is managed by pydal (Python Database Abstraction Layer). `__main__` checks if *pydal* is installed on the host; if not, the program outputs a message and closes.

The module will also check if the host is connected to the internet, this is required because the data used to compute portfolios is pulled from an API. If the host is not connected to the internet the program will close.

The connectivity test is performed by trying to open a socket on google's TCP port 80.

This module also clears the console of whatever was currently displayed on it prior starting the console.

CHAPTER 2

Database Layer

__db__ is used to handle the database in which the application is storing operations, tickers, portfolios and reports.

The layer to manage the database is provided by *pyDal*, this is an open source python package acting as an abstraction layer between an application and a database.

pyDal can handle multiple types of database, in this case since there will not be intensive database operations we use sqlite3.

When imported the module is checking if the *db* folder (located at the same level than the application folder) is present. The folder is created if not found.

The database is then initialize with the following statement:

```
folder = os.getcwd()+('/db' if os.name != 'nt' else '\db')
db = DAL('sqlite://storage.sqlite', folder=folder, migrate_enabled=True)
```

Where *folder* is the database location and *migrate_enabled* indicates to *pyDal* than it should not assume that the tables exists, but rather check and compare against the *define_table* statements.

For this project we defined the following tables:

Investments - *investments*: Contains the investments made by the user, each entry features a *date*, a *name* a volatility based portfolio *vol*, a momentum based portfolio *cmr*, a composite based portfolio *cmp* and a register containing the prices of all the securities listed in each portfolio (the portfolios and the price register are stored in *json*).

Portfolios - *portfolios*: Contains the computed and ranked portfolios that the user chose to save after computing. Each entry features a *date*, a *name* a volatility based portfolio *vol*, a momentum based portfolio *cmr*, a composite based portfolio *cmp* and a register containing the prices of all the securities listed in each portfolio (the portfolios and the price register are stored in *json*).

Symbols - *symbols*: Contains the tickers and their corresponding names, each entry features a ticker *ticker* and a name *name*.

Checks - *checks*: Contains the reports of comparisons of portfolios (old vs now) generated by users and saved in the database for future usage. Each entry features a name *name*, a volatility based portfolio *vol*, a momentum based portfolio *cmr*, a composite based portfolio *cmp* and date *date*. (the portfolios are stored in *json*).

The *json* structure for *checks* table is as follow:

```
{
"ACB": {
    "abs": -109.92,
    "date": "2019-04-22 16:57:42",
    "new": 9.04,
    "old": 9.1,
    "qty": 1832,
    "rel": -0.66
},
"AMD": {
    "abs": -30.81,
    "date": "2019-04-22 16:57:42",
    "new": 27.88,
    "old": 27.9316,
    "qty": 597,
    "rel": -0.18
},
...
```

CHAPTER 3

Tickers Management

`__tickers__` is handling operations related to tickers passed to the application by the user.

The input of all the operations are dictated by the tickers provided by the user, this module is used to check the files provided by users and to write data into the database.

`tickers.`**`save`** (*ticks*)

Saves the ticker's name into the database and into the global ticker `list`.

Parameters:

- *ticks* : `list` of tickers `string`

The function will check for each ticker if an entry already exists in the database, if not, it will attempt to fetch the name from a web API.
If the fetch is successfull, its result is written into the database to save time next time this ticker is used.
The function also sets the global *TICKERS* and *NAMES* `list` with the tickers and their corresponding names.
When a ticker can not be identified on the web API, it is dicarded and stored in a `list`.
Returns a `list` containing tickers that caused errors.

`tickers.`**`save_portfolio`** (*vol, cmr, cmp, prices, name*)

Writes the computed portfolios into the databse.

Parameters:

- *vol* : `dict` volatility portfolio.
- *cmr* : `dict` momentum portfolio.
- *cmp* : `dict` composite portfolio.
- *prices* : `dict` price register.
- *name* : `string` name of the portfolio.

Saves into the database the items passed as argument.
Returns *True* if it succeeded.

`tickers.`**`validate`** (*raw*)

Checks the content of the data provided by the user.
The user provides tickers to the application by writing them into a file that is loaded through the console interface with the <load> command.
We expect the file to be filled with coma separated tickers `string`.
It is necessary to check that the syntax of the file is correct prior passing this data further into the application to avoid errors.

Parameters:

- *raw* : `string` content of the user provided file.

The function strips the raw data from spaces, carrier returns and split the content around comas. It will also check if there was a trailing coma or if the user mistakenly put two comas instead of one.

Returns a `list` of sanitized tickers

CHAPTER 4

API Management

__`api`__ handles communications between the host and the web APIs.

A web API is an Application Programming Interface for a web server or a web browser. On the server side, it consists or one or more publicly exposed endpoints accessible through *http* requests. The content is typically expressed in JSON (JavaScript Object Notation) or XML (eXtensible Markup Language).

__`smartbetas`__ relies on two APIs. One is used to convert tickers to company names (As example: AAPL -> Apple Inc), the second one is used to get information about a security (daily open, daily high, daily low, daily close and daily volume).

The ticker API is totally free and open source, it is maintained by a single contributor (Kambala Yashwanth), this API simply takes a ticker as input and returns the name of the equity if found.

(Alpha Vantage) provides the stock data API. This is not a free service, however they offer a free limited access to their data. The following constraints are applicable when using a free account:

- 5 API requests per minute
- 500 API requests per day

Note that Alpha Vantage requires free users to register to get an API key. The key is stored in __`gbl`__.

`api.`**`symbol`** (*tick*)

Determines the name of a security based on a ticker.

Parameters:

- *tick* : `string` security ticker.

Constructs the request URL with the ticker passed as argument then makes the request and parses the data received from the API.
Data sample from the API: `[{"symbol":"AAPL","name":"Apple Inc."}]`
This function is always called before the software tries to obtain stock data about a security, therefore it filters wrong/bad tickers provided by the user.
Returns a `string` contraining the name of the security OR `None` if the API failed to return data for the ticker.

`api.`**`tsd`** (*tick*)

Fetches a security daily data from 20 years ago till now.

Parameters:

- *tick* : `string` security ticker.

The function makes the API call and parses the JSON, then to assert that data was indeed returned, it checks the length of the data: - l=2 : Data returned - l<2 : No data returned
If no data is returned, the function assumes that the maximum amount of calls per minute is

reached; it will wait for 15 seconds and retry if it fails again it will wait again etc.. till data is returned.
Note: The function can not handle the maximum amount of requests, if it is the case, it will loop forever.
Data sample:

```
{
"Meta Data": {
    "1. Information": "Daily Prices (open, high, low, close) Volumes",
    "2. Symbol": "AAPL",
    "3. Last Refreshed": "2019-04-26",
    "4. Output Size": "Full size",
    "5. Time Zone": "US/Eastern"
},
"Time Series (Daily)": {
    "2019-04-26": {
        "1. open": "204.9000",
        "2. high": "205.0000",
        "3. low": "202.1200",
        "4. close": "204.3000",
        "5. volume": "18611948"
    },
    ...
```

Once the data was obtained, the function will create a `list` of tuple containing in each position a datetime object and the closing price of the security. (List starts with the date closest to now)
List sample : `[(datetime.datetime(2019, 4, 18, 0, 0), 203.86)]`
Returns a `list` of `tuple` with date and closing price.

Computing results

__compute__ is used to compute volatility, momentum and composite based porfolio.

This module gets from the stock data API the security closing value and computes for each security the volatility and the momentum.

Based on the security ranking it also computes a composite portfolio.

To expedite things, this module uses python's *concurrent.futures* to fetch data from the API in a parallel fashion. This is done to save time as *HTTP* requests take time no matter how poweful the host running the code is.

`compute.`**`compute`** (*tick*)

Returns the volatility, momentum and price of a ticker.
This function gets the tickers data with the `api.tsd()`, the volatility with `volatility.volability()` and the momentum with the module `cumulative.ret()`.
These function calls are segregated into this function to enable parallel processing.
Returns a `list` containing the volatility, the momentum and the price of a ticker.

`compute.`**`smartbetas`** (*tickers*)

Builds three portfolios based on volatility, momentum and a composite score.
Parameters:

- *tickers* : `list` of tickers to compute

Uses `ProcessPoolExecutor` to perform for each ticker the following tasks in parallel (using the function `compute.compute()`):

- Get data from the web API - `api.tsd()`
- Compute volatility - `volatility.volatility()`
- Compute momentum - `cumulative.ret()`

The output of the parallel processing is stored in a `list` containing for each ticker a `list` of `tuples`.
Sample:

```
[[('t1', volatility), ('t1', momentum), ('t1', price)],
    [('t2', vol), ('t2', vol), ('t2', vol)]]
```

The volability portfolio is built by sorting the `list` in an ascending fashion and extracting only the tickers and their volatility.
The momentum portfolio is built by sorting the `list` in a descending fashion and extracting the tickers and their momentum.
The prices are also extracted to allow for future comparisons against the market prices.
The composite portfolio is computed with `composite.composite()`.
The results are stored in the global variables.
Returns nothing

5.1 Volatility

__`volatility`__ is used to compute the volatility of a security.

In this application, momentum is considered on a period between two years ago and now.

To calculate the volatility, a standard deviation needs to be performed. To do so we use the standard python's statistics package.

`volatility.`**`volatility`** (*data*)

Computes the volatility of a security.

Parameters:

- *data* : `list` of `tuples`

Note: The function argument is built by *`api.tsd()`*, it is structured as follow: `[(datetime.datetime(2019, 4, 18, 0, 0), 203.86)]`.
The function uses __`datetime`__ to determine the dates between which the volatility shall be calculated. A new `list` containing only the values to consider (the ones within the right time frame). The constructed list is then passed to *statistics.stdev* to compute the deviation.
Returns a the volatility of the security (`float`)

5.2 Momentum

__`cumulative`__ is used to compute cumulative returns; also known as momentum.

In this application, momentum is considered as the returns on a period between 12 months ago and 2 months ago.

All the date and time operations are handled with python's standard *datetime* module.

`cumulative.`**`momentum`** (*vector*)

Computes the momentum of a vector.

Parameters:

- *vector* : `list` of `floats`

The function computes the returns of the data passed as argument:

- momentum = 100*(new - old)/old

Returns the momentum of the vector (`float`)

`cumulative.`**`ret`** (*data*)

Computes the cumulative returns (momentum) of a security.

Parameters:

- *data* : `list` of `tuples`

Note: The function argument is built by *`api.tsd()`*, it is structured as follow: `[(datetime.datetime(2019, 4, 18, 0, 0), 203.86)]`.
The function uses __`datetime`__ to determine the dates between which the momentum shall be calculated. A new `list` containing only the values to consider (the ones within the right time frame) is built and passed to *`cumulative.momentum()`*.
Returns a the momentum of the security (`float`)

5.3 Composite

`__composite__` is used to determine a composite portfolio.

In this application we define a composite portfolio as a portfolio based on the ranking of a security in the volatility portfolio and in the momentum portfolio.

The score of security is the sum of its index (position) in the two other portfolio. The securities are then sorted in an ascending fasion.

`composite.`**`composite`** (*vol*, *cmr*)

Ranks securities in a composite fashion.

Parameters:

- *vol* : `dict` volatility portfolio.
- *cmr* : `dict` momentum portfolio.

Note: at this point, the two portfolios have the same tickers, only their ranking varies.
The function builds a `dict` with the tickers and set their score to zero; sample {'ticker': 0}. Then it adds to the ticker score their index in volatility and momentum portfolio.

To rank the tickers, the `dict` is transformed into a `tuple`,
sorted and transformed back into a `dict`.
Returns a `dict` containing tickers and their score.

Investing

__money__ is handle investments in the computed portfolios.

In the application, when a user requested to compute portfolios based on the tickers he provided, he can choose to invest money in the portfolios to be able to check them later on.

For now one can only invest 100'000 USD in each portfolio with a reparatition of `1/n` where `n` is the amount of tickers in a portfolio.

`money.`**`invest`** (*p_size*, *p_vol*, *p_cmr*, *p_cmp*, *prices*, *s_name*, *stack=100000*)

Invests 100'000 USD in each computed portfolio.

Parameters:

- *p_size* : `int` amount of tickers per portfolio.
- *p_vol* : `dict` volatility portfolio.
- *p_cmr* : `dict` momentum portfolio.
- *p_cmp* : `dict` composite portfolio.
- *prices* : `dict` price register.
- *s_name* : `string` name of the investment.
- *stack* : `int` defaults to 100000, money to invest.

For each portfolio, the function determines to max amount of shares that can be purchased and builds a `dict` of tickers and quantity of shares.
The function then stores the outcomes into the database, the prices of the securities are also stored for future usage.
Returns nothing.

CHAPTER 7

User Interface

`console` is used to have a neat console interface.

The user interface is built on top of python's standard *cmd* module.

The console interface inherits pretty much everything from the standard cmd *class*. The following parameters are modified:

- *intro* : 'Smart Betas Investing - Requires an internet connection'
- *prompt* : (beta)

A new parameter is introduced:

- 'out' : '–>', this string preceeds outputs on the console

class `console.`**`betacmd`** (*completekey='tab'*, *stdin=None*, *stdout=None*)

`do_bye` (*arg*)
Terminates the program.
usage: bye

`do_check` (*arg*)
Generate a short report detailling returns for an investment
usage: check

`do_compute` (*arg*)
Computes tickers in the pipe into a portfolio
usage: compute

`do_invest` (*arg*)
Invest 100'000 USD into each of the calculated portfolio.
usage: invest

`do_load` (*arg*)
Loads a file containing tickers
usage: load <FILENAME>
Tickers in the file should be coma separated.

`do_portfolios` (*arg*)
Displays stored portfolios
Users can select a portfolio, view its securities and/or invest in it.
usage: portfolios

`do_report` (*arg*)
Displays previously generated reports

usage: report

`do_show` (*arg*)
Displays information to the user.
tickers: shows the list of tickers loaded to build a portfolio.
portfolio: shows the session portfolio (after <compute>)
invest: shows the investment sessions performed by the user.
usage: show <tickers/portfolio/invest>

`do_symbols` (*arg*)
Display tickers stored in the database
usage: symbols

`emptyline` ( )
Called when an empty line is entered in response to the prompt.
If this method is not overridden, it repeats the last nonempty command entered.

Checking Out Results

__check__ is used to determine the returns of an investment.

This module fetches from the stock data API today's securities values and compares them with the data from a previous investment session stored in the database. (API data provided through `api.tsd()`)

To expedite things, this module uses python's *concurrent.futures* to fetch data from the API in a parallel fashion. This is done to save time as *HTTP* requests take time no matter how poweful the host running the code is.

`check.`**`get_price`** (*tick*)

Gets the current price of a security.

Parameters:

- *tick* : `string` security ticker

Fetches the latest price of the ticker from the stock api. The result is stored in a `tuple` : `` `('ticker', 149) ``
Returns a `tuple` containing the ticker and its latest price.

`check.`**`returns`** (*inv, pfl, prices*)

Computes the return of a portfolio.
Parameters:

- *inv* : `list` past investment session row from the db
- *pfl* : `string` name of the portfolio
- *prices* : `dict` latest investment's ticker prices

Computes the absolute change and the returns for each ticker in the portfolio. These values are calculated as follow:

- Absolute change : (qty *new) - (qty * old) (*rounded at 2 digits*)
- Returns : [(new-old)/old*100] (*rounded at 2 digits*)

The total absolute change and return is handled by `check.t_returns()`.
Returns a `dict` containing for each ticker the initial price, the new price, the absolute change, the returns, the quantity of shares and the date of the purchase.

`check.`**`sessions`** (*id*)

Determines returns of an investment session.
Parameters:

- *id* : `integer` investment session database id.

Uses `ProcessPoolExecutor` to request in parallel the latest prices of each tickers using

`check.get_price()` and stores the results in a `dict` : `{'ticker1': 100, 'ticker2': 200}`.
The returns of the portfolios are calculated by *`check.returns()`*.
Once the returns are calculated, the results are saved in the database, this is to allow users to consults the results later on without having the re-compute the returns or query any data from the API.

Returns:

- *r_vol* : `dict` Returns of the volability portfolio.
- *r_cmr* : `dict` Returns of the momentum portfolio.
- *r_cmp* : `dict` Returns of the composite portfolio.
- *inv['name']* : `string` investment session name.

`check.`**`t_returns`**(*inv*, *pfl*, *prices*, *date*)
Computes the total return of a portfolio.
Parameters:

- *inv* : `list` past investment session row from the db
- *pfl* : `string` name of the portfolio
- *prices* : `dict` latest investment's ticker prices
- *date* : `string` date of the purchase

Computes the sum of the initial portfolio value and the sum of the current portfolio value. Based on the sums, it computes the absolute change and the returns using the same formula as in *`check.returns()`*.
Returns a `dict` containing the total initial price, the new price, the absolute change, the returns and the date of the purchase.

CHAPTER 9

Input / Outputs

__i_o__ is handle inputs and outputs generated from the console by the user.

`i_o.`**`investment`** (*qty*)

Prompts instructions to guide a user to invest 100'000 USD into a portfolio.

```
(beta): invest
 Invest 100'000 USD in each portefolio (y/n): y
 Invest in the [x] top securities (1): 1
 Name of the investment : Glorious Investment
```

Returns nothing.

`i_o.`**`portfolio`** (*vol, cmr, cmp, prices*)

Displays a computed portfolio.

```
-----------------------------------------------------------------------
| Pos |      Volatility      |       Momentum       |      Composite       |
-----------------------------------------------------------------------
|  1  |    AAPL      21.98   |    AAPL      5.88 %  |    AAPL     204.3 $  |
|  2  |    TSLA      29.98   |    TSLA      0.3 %   |    TSLA     235.14 $ |
-----------------------------------------------------------------------
```

Returns nothing.

`i_o.`**`portfolios`** (*p_flo*)

Displays a table listing all the saved portfolios.

```
-----------------------------------------------------------------------
|  id |   Date    |       Name      |              Tickers              |
-----------------------------------------------------------------------
|  1  |2019-04-23|   Apple Test   |                AAPL               |
|  2  |2019-04-23|   Test Apple   |                AAPL               |
|  3  |2019-04-23|      Test      |                AAPL               |
|  4  |2019-04-23| 14 Tickers ftw|      AMD, ACB, GWW, GOOGL...       |
|  5  |2019-04-27|   Apple Only   |                AAPL               |
|  6  |2019-04-27|  Apple Tesla   |           AAPL, TSLA...           |
-----------------------------------------------------------------------
```

Returns nothing.

`i_o.`**`reports`** (*rep*)

Displays a listing all the previously saved reports.

```
-----------------------------------------------------------------------
| Pos |     Id     |            Date            |            Name            |
-----------------------------------------------------------------------
|  1  |     1      |         2019-04-26         |        6 Big Stuffs        |
|  2  |     2      |         2019-04-27         |        6 Big Stuffs        |
|  3  |    3     |        2019-04-27        |         Test         |
|  4  |    4     |        2019-04-27        |      6 Big Stuffs     |
|  5  |    5     |        2019-04-28        |        One One        |
----------------------------------------------------------------------
```

Returns nothing.

i_o.**returns** (*vol, cmr, cmp, name*)
Displays a report with the returns for each portfolio.

```
--------------------------------------------------------------------------------
                              One One - Report
--------------------------------------------------------------------------------
                        Volatility Based Portfolio
--------------------------------------------------------------------------------
| Ticker |N Shares| Purchase Date |  Initial   |  Current   | Abs Change|Returns
 |
--------------------------------------------------------------------------------
|  MBRX  | 12270  |   2019-04-27  |   1.63 $   |   1.42 $   | -2576.7 $ |-12.88
%|
|  RAD   |  1992  |   2019-04-27  | 10.0401 $ |   9.08 $   | -1912.52 $|-9.56 %
 |
|  KEYW  |  1784  |   2019-04-27  |  11.21 $   |   11.3 $   |  160.56 $ | 0.8 %
 |
|  ACB   |  2212  |   2019-04-27  |  9.0396 $ |   9.04 $   |   0.88 $  | 0.0 %
 |
|  STLD  |  609   |   2019-04-27  |  32.86 $   |  31.58 $   | -779.52 $ | -3.9 %
 |
| Total  |   NA   |   2019-04-27  | 100006.0 $| 94898.7 $ | -5107.3 $ |-5.11 %
 |
--------------------------------------------------------------------------------
                         Momentum Based Portfolio
--------------------------------------------------------------------------------
| Ticker |N Shares| Purchase Date |  Initial   |  Current   | Abs Change|Returns
 |
--------------------------------------------------------------------------------
|  AMD   |  717   |   2019-04-27  |  27.89 $   |  27.88 $   |  -7.17 $  |-0.04 %
 |
|  ACB   |  2212  |   2019-04-27  |  9.0396 $ |   9.04 $   |   0.88 $  | 0.0 %
 |
|  GWW   |   69   |   2019-04-27  | 291.015 $ | 291.91 $ |  61.76 $  | 0.31 %
 |
| GOOGL  |   16   |   2019-04-27  | 1264.28 $ | 1277.42 $ |  210.24 $ | 1.04 %
 |
|  BABA  |  107   |   2019-04-27  | 186.7425 $| 187.09 $ |  37.18 $  | 0.19 %
 |
| Total  |   NA   |   2019-04-27  | 100282.7 $| 100585.6 $|  302.9 $  | 0.3 %
 |
--------------------------------------------------------------------------------
                        Composite Based Portfolio
--------------------------------------------------------------------------------
| Ticker |N Shares| Purchase Date |  Initial   |  Current   | Abs Change|Returns
 |
--------------------------------------------------------------------------------
|  ACB   |  2212  |   2019-04-27  |  9.0396 $ |   9.04 $   |   0.88 $  | 0.0 %
 |
|  AMD   |  717   |   2019-04-27  |  27.89 $   |  27.88 $   |  -7.17 $  |-0.04 %
 |
|  MBRX  | 12270  |   2019-04-27  |   1.63 $   |   1.42 $   | -2576.7 $ |-12.88
%|
|  KEYW  |  1784  |   2019-04-27  |  11.21 $   |   11.3 $   |  160.56 $ | 0.8 %
 |
|  BABA  |  107   |   2019-04-27  | 186.7425 $| 187.09 $ |  37.18 $  | 0.19 %
 |
| Total   |   NA    |   2019-04-27   | 99972.9 $ | 97587.7 $ | -2385.2 $ |-2.39 %
 |
--------------------------------------------------------------------------------
```

Returns nothing.

i_o.**sessions** (*inv*)

Displays a table listing all the investments made by the user.

```
-----------------------------------------------------------------------
| Pos |     Id     |             Date          |            Name         |
-----------------------------------------------------------------------
|  1  |     1      |   2019-04-22 16:57:42     |        6 Big Stuffs      |
|  2  |     2      |   2019-04-27 13:59:42     |           Test           |
|  3  |     3      |   2019-04-27 14:05:07     |          One One         |
|  4  |     4      |   2019-04-28 12:31:58     |   Glorious Investment    |
|  5  |     5      |   2019-04-28 12:32:38     |   Glorious Investment    |
-----------------------------------------------------------------------
```

Returns nothing.

i_o.**tickers** (*row*)

Lists all the tickers and names saved in the database.

```
(beta): symbols
-->    0: AAPL -  Apple Inc.
-->    1: TSLA -  Tesla Inc.
-->    2: GOOGL -  Alphabet Inc.
-->    3: BABA -  Alibaba Group Holding Limited
-->    4: GWW -  W.W. Grainger Inc.
-->    5: CVX -  Chevron Corporation
-->    6: HAL -  Bank Nova Scotia Halifax Pfd 3
-->    7: RAD -  Credit Suisse X-Links Monthly Pay 2xLeveraged Alerian MLP
Index Exchange Traded Notes due May 16 2036
-->    8: MBRX -  Moleculin Biotech Inc.
-->    9: KEYW -  The KEYW Holding Corporation
-->   10: BBBY -  Bed Bath & Beyond Inc.
-->   11: AMD -  Advanced Micro Devices Inc.
-->   12: STLD -  Steel Dynamics Inc.
-->   13: ACB -  Aurora Cannabis Inc.
-->   14: ATHEN -  Athene Holding Ltd. Class A
```

Returns nothing.

CHAPTER 10

Global Variables

`__gbl__` is used to list/access/set variables with a global scope.

In this application we need to make some chunks of data available for multiple modules, for this reason we use variables of a global scope. For the sake of clarity, they are declared on a module dedicated to that.

The following global variables are declared:

- *TICKERS* : `list` storing tickers to compute
- *NAMES* : `list` storing *TICKERS*'s name
- *P_VOL* : `dict` storing the volatility portfolio
- *P_CMR* : `dict` storing the momentum portfolio
- *P_CMP* : `dict` storing the composite portfolio
- *PRICES* : `dict` storing the prices register
- *API_KEY* : `string` storing the API key
- *API_URL* : `string` storing the API URL

Test Modules

`__test_cumulative__` tests the function that computes the volatility of a vector in *`volatility.volatility()`*

This module can be exectuted with `python test_volatility.py`

class `test_volatility.`**`TestVolatility`** (*methodName='runTest'*)

`__test_cumulative__` tests the function that computes the momentum of a vector in *`cumulative.momentum()`*

This module can be exectuted with `python test_cumulative.py`

class `test_cumulative.`**`TestVolatility`** (*methodName='runTest'*)

- *Index*
- *Module Index*
- *Search Page*

Python Module Index