project_1_starter

January 30, 2021

1 Project 1: Trading with Momentum

1.1 Instructions

Each problem consists of a function to implement and instructions on how to implement the function. The parts of the function that need to be implemented are marked with a `# TODO` comment. After implementing the function, run the cell to test it against the unit tests we've provided. For each problem, we provide one or more unit tests from our `project_tests` package. These unit tests won't tell you if your answer is correct, but will warn you of any major errors. Your code will be checked for the correct solution when you submit it to Udacity.

1.2 Packages

When you implement the functions, you'll only need to you use the packages you've used in the classroom, like Pandas and Numpy. These packages will be imported for you. We recommend you don't add any import statements, otherwise the grader might not be able to run your code.

The other packages that we're importing are `helper`, `project_helper`, and `project_tests`. These are custom packages built to help you solve the problems. The `helper` and `project_helper` module contains utility functions and graph functions. The `project_tests` contains the unit tests for all the problems.

1.2.1 Install Packages

```
In [10]: import sys
         !{sys.executable} -m pip install -r requirements.txt
```

```
Requirement already satisfied: colour==0.1.5 in /opt/conda/lib/python3.6/site-packages (from -r
Collecting cvxpy==1.0.3 (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/a1/59/2613468ffbbe3a818934d06b81b9f4877fe0
    100% || 880kB 11.3MB/s ta 0:00:01
Requirement already satisfied: cycler==0.10.0 in /opt/conda/lib/python3.6/site-packages/cycler-0
Collecting numpy==1.13.3 (from -r requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/57/a7/e3e6bd9d595125e1abbe162e323fd2d06f6f
    100% || 17.0MB 1.9MB/s eta 0:00:01  7% |                              | 1.2MB 14.0MB/s eta 0:
Collecting pandas==0.21.1 (from -r requirements.txt (line 5))
  Downloading https://files.pythonhosted.org/packages/3a/e1/6c514df670b887c77838ab856f57783c07e8
    100% || 26.2MB 1.6MB/s eta 0:00:01  5% |                              | 1.5MB 21.2MB/s eta 0
Collecting plotly==2.2.3 (from -r requirements.txt (line 6))
```

```
  Downloading https://files.pythonhosted.org/packages/99/a6/8214b6564bf4ace9bec8a26e7f89832792be
    100% || 1.1MB 14.3MB/s ta 0:00:01    84% |     | 911kB 19.7MB/s eta 0:00:01
Requirement already satisfied: pyparsing==2.2.0 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: python-dateutil==2.6.1 in /opt/conda/lib/python3.6/site-packages
Requirement already satisfied: pytz==2017.3 in /opt/conda/lib/python3.6/site-packages (from -r r
Requirement already satisfied: requests==2.18.4 in /opt/conda/lib/python3.6/site-packages (from
Collecting scipy==1.0.0 (from -r requirements.txt (line 11))
  Downloading https://files.pythonhosted.org/packages/d8/5e/caa01ba7be11600b6a9d39265440d7b3be3d
    100% || 50.0MB 701kB/s ta 0:00:011 0% |                                  | 71kB 14.9MB/s eta 0
Requirement already satisfied: scikit-learn==0.19.1 in /opt/conda/lib/python3.6/site-packages (f
Requirement already satisfied: six==1.11.0 in /opt/conda/lib/python3.6/site-packages (from -r re
Collecting tqdm==4.19.5 (from -r requirements.txt (line 14))
  Downloading https://files.pythonhosted.org/packages/71/3c/341b4fa23cb3abc335207dba057c790f3bb3
    100% || 61kB 11.2MB/s ta 0:00:01
Collecting osqp (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/76/82/b0693a167e4b9b5e94f4988f6df3d7866e9e
    100% || 215kB 15.2MB/s ta 0:00:01
Collecting ecos>=2 (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/55/ed/d131ff51f3a8f73420eb1191345eb49f269f
    100% || 153kB 8.5MB/s ta 0:00:01    48% |                | 71kB 14.9MB/s eta 0:00:01
Collecting scs>=1.1.3 (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/1a/72/33be87cce255d4e9dbbfef547e9fd6ec7ee9
    100% || 3.6MB 5.8MB/s eta 0:00:01
Collecting multiprocess (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/8f/dc/426a82723c460cfab653ebb717590103d6e3
    100% || 102kB 12.7MB/s a 0:00:01
Requirement already satisfied: fastcache in /opt/conda/lib/python3.6/site-packages (from cvxpy==
Requirement already satisfied: toolz in /opt/conda/lib/python3.6/site-packages (from cvxpy==1.0.
Requirement already satisfied: decorator>=4.0.6 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: nbformat>=4.2 in /opt/conda/lib/python3.6/site-packages (from plo
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/python3.6/site-packages (
Requirement already satisfied: idna<2.7,>=2.5 in /opt/conda/lib/python3.6/site-packages (from re
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /opt/conda/lib/python3.6/site-packages (
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.6/site-packages (fro
Collecting qdldl (from osqp->cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/ec/a3/db0e7c9fec5387dc33cbd2819329c141ba76
    100% || 71kB 10.4MB/s ta 0:00:01
Collecting dill>=0.3.3 (from multiprocess->cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/52/d6/79f40d230895fa1ce3b6af0d22e0ac79c651
    100% || 81kB 10.5MB/s ta 0:00:01
Requirement already satisfied: jupyter-core in /opt/conda/lib/python3.6/site-packages (from nbfo
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /opt/conda/lib/python3.6/site-packages
Requirement already satisfied: traitlets>=4.1 in /opt/conda/lib/python3.6/site-packages (from nb
Building wheels for collected packages: cvxpy, plotly, scs, qdldl
  Running setup.py bdist_wheel for cvxpy ... done
  Stored in directory: /root/.cache/pip/wheels/2b/60/0b/0c2596528665e21d698d6f84a3406c52044c7b4c
  Running setup.py bdist_wheel for plotly ... done
  Stored in directory: /root/.cache/pip/wheels/98/54/81/dd92d5b0858fac680cd7bdb8800eb26c001dd9f5
  Running setup.py bdist_wheel for scs ... done
  Stored in directory: /root/.cache/pip/wheels/df/d0/79/37ea880586da03c620ca9ecd5e42adbd86bc6ea8
  Running setup.py bdist_wheel for qdldl ... done
  Stored in directory: /root/.cache/pip/wheels/a9/77/d6/726fc4a2ae1513b4663b81721f5d75e9b4fe9d74
Successfully built cvxpy plotly scs qdldl
tensorflow 1.3.0 requires tensorflow-tensorboard<0.2.0,>=0.1.0, which is not installed.
moviepy 0.2.3.2 has requirement tqdm==4.11.2, but you'll have tqdm 4.19.5 which is incompatible.
Installing collected packages: numpy, scipy, qdldl, osqp, ecos, scs, dill, multiprocess, cvxpy,
  Found existing installation: numpy 1.12.1
    Uninstalling numpy-1.12.1:
      Successfully uninstalled numpy-1.12.1
  Found existing installation: scipy 1.2.1
    Uninstalling scipy-1.2.1:
      Successfully uninstalled scipy-1.2.1
  Found existing installation: dill 0.2.7.1
    Uninstalling dill-0.2.7.1:
      Successfully uninstalled dill-0.2.7.1
  Found existing installation: pandas 0.23.3
    Uninstalling pandas-0.23.3:
      Successfully uninstalled pandas-0.23.3
  Found existing installation: plotly 2.0.15
    Uninstalling plotly-2.0.15:
      Successfully uninstalled plotly-2.0.15
  Found existing installation: tqdm 4.11.2
    Uninstalling tqdm-4.11.2:
      Successfully uninstalled tqdm-4.11.2
Successfully installed cvxpy-1.0.3 dill-0.3.3 ecos-2.0.7.post1 multiprocess-0.70.11.1 numpy-1.13
```

1.2.2 Load Packages

```
In [1]: import pandas as pd
        import numpy as np
        import helper
        import project_helper
        import project_tests
```

1.3 Market Data

1.3.1 Load Data

The data we use for most of the projects is end of day data. This contains data for many stocks, but we'll be looking at stocks in the S&P 500. We also made things a little easier to run by narrowing down our range of time period instead of using all of the data.

```
In [3]: df = pd.read_csv('../../data/project_1/eod-quotemedia.csv', parse_dates=['date'], index_

        close = df.reset_index().pivot(index='date', columns='ticker', values='adj_close')
        print('Loaded Data')
```

```
Loaded Data
```

1.3.2 View Data

Run the cell below to see what the data looks like for `close`.

```
In [4]: close.head()
        #project_helper.print_dataframe(close)
```

```
Out[4]: ticker               A         AAL         AAP        AAPL        ABBV  \
        date
        2013-07-01 29.99418563 16.17609308 81.13821681 53.10917319 34.92447839
        2013-07-02 29.65013670 15.81983388 80.72207258 54.31224742 35.42807578
        2013-07-03 29.70518453 16.12794994 81.23729877 54.61204262 35.44486235
        2013-07-05 30.43456826 16.21460758 81.82188233 54.17338125 35.85613355
        2013-07-08 30.52402098 16.31089385 82.95141667 53.86579916 36.66188936

        ticker             ABC         ABT         ACN        ADBE         ADI  \
        date
        2013-07-01 50.86319750 31.42538772 64.69409505 46.23500000 39.91336014
        2013-07-02 50.69676639 31.27288084 64.71204071 46.03000000 39.86057632
        2013-07-03 50.93716689 30.72565028 65.21451912 46.42000000 40.18607651
        2013-07-05 51.37173702 31.32670680 66.07591068 47.00000000 40.65233352
        2013-07-08 52.03746147 31.76628544 66.82065546 46.62500000 40.25645492

        ticker         ...              XL        XLNX         XOM        XRAY  \
        date           ...
        2013-07-01     ...     27.66879066 35.28892781 76.32080247 40.02387348
        2013-07-02     ...     27.54228410 35.05903252 76.60816761 39.96552964
        2013-07-03     ...     27.33445191 35.28008569 76.65042719 40.00442554
        2013-07-05     ...     27.69589920 35.80177117 77.39419581 40.67537968
        2013-07-08     ...     27.98505704 35.20050655 77.96892611 40.64620776

        ticker             XRX         XYL         YUM         ZBH        ZION  \
        date
        2013-07-01 22.10666494 25.75338607 45.48038323 71.89882693 27.85858718
        2013-07-02 22.08273998 25.61367511 45.40266113 72.93417195 28.03893238
        2013-07-03 22.20236479 25.73475794 46.06329899 72.30145844 28.18131017
        2013-07-05 22.58516418 26.06075017 46.41304845 73.16424628 29.39626730
        2013-07-08 22.48946433 26.22840332 46.95062632 73.89282298 29.57661249

        ticker             ZTS
        date
        2013-07-01 29.44789315
        2013-07-02 28.57244125
```

```
2013-07-03 28.16838652
2013-07-05 29.02459772
2013-07-08 29.76536472

[5 rows x 495 columns]
```

1.3.3 Stock Example

Let's see what a single stock looks like from the closing prices. For this example and future display examples in this project, we'll use Apple's stock (AAPL). If we tried to graph all the stocks, it would be too much information.

```
In [5]: apple_ticker = 'AAPL'
        project_helper.plot_stock(close[apple_ticker], '{} Stock'.format(apple_ticker))
```

1.4 Resample Adjusted Prices

The trading signal you'll develop in this project does not need to be based on daily prices, for instance, you can use month-end prices to perform trading once a month. To do this, you must first resample the daily adjusted closing prices into monthly buckets, and select the last observation of each month.

Implement the `resample_prices` to resample `close_prices` at the sampling frequency of `freq`.

```
In [6]: def resample_prices(close_prices, freq='M'):
            """
            Resample close prices for each ticker at specified frequency.

            Parameters
            ----------
            close_prices : DataFrame
                Close prices for each ticker and date
            freq : str
                What frequency to sample at
                For valid freq choices, see http://pandas.pydata.org/pandas-docs/stable/timeseri

            Returns
            -------
            prices_resampled : DataFrame
                Resampled prices for each ticker and date
            """
            # TODO: Implement Function
            #df.apply(np.sum, axis=0)
            result = close_prices.resample('M').last()
            return result

        project_tests.test_resample_prices(resample_prices)
```

```
Tests Passed
```

1.4.1 View Data

Let's apply this function to `close` and view the results.

```
In [7]: monthly_close = resample_prices(close)
        project_helper.plot_resampled_prices(
            monthly_close.loc[:, apple_ticker],
            close.loc[:, apple_ticker],
            '{} Stock - Close Vs Monthly Close'.format(apple_ticker))
```

1.5 Compute Log Returns

Compute log returns (R_t) from prices (P_t) as your primary momentum indicator:

$$R_t = log_e(P_t) - log_e(P_{t-1})$$

Implement the `compute_log_returns` function below, such that it accepts a dataframe (like one returned by `resample_prices`), and produces a similar dataframe of log returns. Use Numpy's log function to help you calculate the log returns.

```
In [8]: def compute_log_returns(prices):
            """
            Compute log returns for each ticker.

            Parameters
            ----------
            prices : DataFrame
                Prices for each ticker and date

            Returns
            -------
            log_returns : DataFrame
                Log returns for each ticker and date
            """
            # TODO: Implement Function

            retDF = prices /prices.shift(1)
            print(retDF.head(3))
            print(retDF.shape)
            #res = retDF
            res = retDF.apply(lambda x : np.log(x), axis=1)

            print(res.head(3))
            print(res.shape)
            return res

        project_tests.test_compute_log_returns(compute_log_returns)
```

```
                  FTM       TUEL          VZQD       NPHI         WOS
2008-08-31        nan        nan           nan        nan         nan
```

```
2008-09-30 22.91338820 2.06905512 320.13659925 5.90369512 1.04183458
2008-10-31  0.02263717 0.50873100   0.00705274 0.18807161 0.77935353
(4, 5)
                   FTM        TUEL        VZQD        NPHI         WOS
2008-08-31         nan         nan         nan         nan         nan
2008-09-30  3.13172138  0.72709204  5.76874778  1.77557845  0.04098317
2008-10-31 -3.78816218 -0.67583590 -4.95433863 -1.67093250 -0.24929051
(4, 5)
Tests Passed
```

1.5.1 View Data

Using the same data returned from `resample_prices`, we'll generate the log returns.

```
In [9]: monthly_close_returns = compute_log_returns(monthly_close)
        project_helper.plot_returns(
            monthly_close_returns.loc[:, apple_ticker],
            'Log Returns of {} Stock (Monthly)'.format(apple_ticker))
```

```
ticker              A        AAL        AAP       AAPL       ABBV        ABC  \
date
2013-07-31        nan        nan        nan        nan        nan        nan
2013-08-31 1.04270065 0.83514212 0.97066311 1.08377207 0.93689534 0.98403546
2013-09-30 1.10139616 1.17326733 1.03336746 0.97851877 1.04975358 1.07343640

ticker            ABT        ACN       ADBE        ADI    ...             XL  \
date                                                      ...
2013-07-31        nan        nan        nan        nan    ...            nan
2013-08-31 0.90990991 0.97886465 0.96763959 0.94412333    ...     0.94290271
2013-09-30 0.99579958 1.01923875 1.13530055 1.01663786    ...     1.04739847

ticker           XLNX        XOM       XRAY        XRX        XYL        YUM  \
date
2013-07-31        nan        nan        nan        nan        nan        nan
2013-08-31 0.93500350 0.93616296 0.97924440 1.02886598 0.99865223 0.96023039
2013-09-30 1.07901889 0.98715007 1.03542459 1.03677369 1.12711864 1.01956584

ticker            ZBH       ZION        ZTS
date
2013-07-31        nan        nan        nan
2013-08-31 0.94741247 0.94493502 0.97785978
2013-09-30 1.04110609 0.98033607 1.06758148

[3 rows x 495 columns]
(48, 495)
ticker              A          AAL          AAP         AAPL         ABBV  \
date
```

```
2013-07-31        nan         nan         nan         nan         nan
2013-08-31 0.04181412 -0.18015337 -0.02977582  0.08044762 -0.06518370
2013-09-30 0.09657861  0.15979244  0.03282284 -0.02171531  0.04855545

ticker            ABC         ABT         ACN        ADBE         ADI  \
date
2013-07-31        nan         nan         nan         nan         nan
2013-08-31 -0.01609335 -0.09440968 -0.02136190 -0.03289558 -0.05749847
2013-09-30  0.07086509 -0.00420927  0.01905603  0.12689741  0.01650096

ticker         ...              XL        XLNX         XOM        XRAY  \
date           ...
2013-07-31     ...             nan         nan         nan         nan
2013-08-31     ...     -0.05879217 -0.06720501 -0.06596571 -0.02097402
2013-09-30     ...      0.04630944  0.07605219 -0.01293321  0.03481157

ticker            XRX         XYL         YUM         ZBH        ZION  \
date
2013-07-31        nan         nan         nan         nan         nan
2013-08-31 0.02845720 -0.00134868 -0.04058203 -0.05402072 -0.05663911
2013-09-30 0.03611367  0.11966450  0.01937689  0.04028369 -0.01985983

ticker            ZTS
date
2013-07-31        nan
2013-08-31 -0.02238899
2013-09-30  0.06539579

[3 rows x 495 columns]
(48, 495)
```

1.6 Shift Returns

Implement the `shift_returns` function to shift the log returns to the previous or future returns in the time series. For example, the parameter `shift_n` is 2 and `returns` is the following:

```
                           Returns
               A          B          C          D
2013-07-08     0.015      0.082      0.096      0.020      ...
2013-07-09     0.037      0.095      0.027      0.063      ...
2013-07-10     0.094      0.001      0.093      0.019      ...
2013-07-11     0.092      0.057      0.069      0.087      ...
...            ...        ...        ...        ...
```

the output of the `shift_returns` function would be:

```
                        Shift Returns
               A          B          C          D
```

```
2013-07-08     NaN       NaN       NaN       NaN        ...
2013-07-09     NaN       NaN       NaN       NaN        ...
2013-07-10     0.015     0.082     0.096     0.020      ...
2013-07-11     0.037     0.095     0.027     0.063      ...
...            ...       ...       ...       ...
```

Using the same `returns` data as above, the `shift_returns` function should generate the following with `shift_n` as -2:

```
                        Shift Returns
               A         B         C         D
2013-07-08     0.094     0.001     0.093     0.019      ...
2013-07-09     0.092     0.057     0.069     0.087      ...
...            ...       ...       ...       ...        ...
...            ...       ...       ...       ...        ...
...            NaN       NaN       NaN       NaN        ...
...            NaN       NaN       NaN       NaN        ...
```

Note: The "..." represents data points we're not showing.

```
In [10]: def shift_returns(returns, shift_n):
             """
             Generate shifted returns

             Parameters
             ----------
             returns : DataFrame
                 Returns for each ticker and date
             shift_n : int
                 Number of periods to move, can be positive or negative

             Returns
             -------
             shifted_returns : DataFrame
                 Shifted returns for each ticker and date
             """
             # TODO: Implement Function

             returns.shift(shift_n)
             return returns.shift(shift_n)

         project_tests.test_shift_returns(shift_returns)
```

```
Tests Passed
```

1.6.1 View Data

Let's get the previous month's and next month's returns.

```
In [11]: prev_returns = shift_returns(monthly_close_returns, 1)
         lookahead_returns = shift_returns(monthly_close_returns, -1)

         project_helper.plot_shifted_returns(
             prev_returns.loc[:, apple_ticker],
             monthly_close_returns.loc[:, apple_ticker],
             'Previous Returns of {} Stock'.format(apple_ticker))
         project_helper.plot_shifted_returns(
             lookahead_returns.loc[:, apple_ticker],
             monthly_close_returns.loc[:, apple_ticker],
             'Lookahead Returns of {} Stock'.format(apple_ticker))
```

1.7 Generate Trading Signal

A trading signal is a sequence of trading actions, or results that can be used to take trading actions. A common form is to produce a "long" and "short" portfolio of stocks on each date (e.g. end of each month, or whatever frequency you desire to trade at). This signal can be interpreted as rebalancing your portfolio on each of those dates, entering long ("buy") and short ("sell") positions as indicated.

Here's a strategy that we will try: > For each month-end observation period, rank the stocks by *previous* returns, from the highest to the lowest. Select the top performing stocks for the long portfolio, and the bottom performing stocks for the short portfolio.

Implement the `get_top_n` function to get the top performing stock for each month. Get the top performing stocks from `prev_returns` by assigning them a value of 1. For all other stocks, give them a value of 0. For example, using the following `prev_returns`:

```
                                     Previous Returns
               A         B         C         D         E         F         G
2013-07-08     0.015     0.082     0.096     0.020     0.075     0.043     0.074
2013-07-09     0.037     0.095     0.027     0.063     0.024     0.086     0.025
...            ...       ...       ...       ...       ...       ...       ...
```

The function `get_top_n` with `top_n` set to 3 should return the following:

```
                                     Previous Returns
               A         B         C         D         E         F         G
2013-07-08     0         1         1         0         1         0         0
2013-07-09     0         1         0         1         0         1         0
...            ...       ...       ...       ...       ...       ...       ...
```

Note: You may have to use Panda's `DataFrame.iterrows` with `Series.nlargest` in order to implement the function. This is one of those cases where creating a vecorization solution is too difficult.

```
In [12]: def get_top_n(prev_returns, top_n):
             """
             Select the top performing stocks

             Parameters
             ----------
```

```
    prev_returns : DataFrame
        Previous shifted returns for each ticker and date
    top_n : int
        The number of top performing stocks to get

    Returns
    -------
    top_stocks : DataFrame
        Top stocks for each ticker and date marked with a 1
    """
    # TODO: Implement Function
    print(prev_returns.head(2))
    topStocks= prev_returns.copy()
    try:
        for index,row in topStocks.iterrows():
            topColumns = row.nlargest(top_n).index
            topStocks.loc[index] = 0
            # Assign one to top columns
            topStocks.loc[index,topColumns] = 1

        return topStocks.astype(int)
    except TypeError as err:
        print('Error: {}'.format(err))
    return None

project_tests.test_get_top_n(get_top_n)
```

```
            EJGE  SOR  KKXY  DCQR  HZK
2008-08-31   nan  nan   nan   nan  nan
2008-09-30   nan  nan   nan   nan  nan
Tests Passed
```

1.7.1 View Data

We want to get the best performing and worst performing stocks. To get the best performing stocks, we'll use the get_top_n function. To get the worst performing stocks, we'll also use the get_top_n function. However, we pass in -1*prev_returns instead of just prev_returns. Multiplying by negative one will flip all the positive returns to negative and negative returns to positive. Thus, it will return the worst performing stocks.

```
In [13]: top_bottom_n = 50
         df_long = get_top_n(prev_returns, top_bottom_n)
         df_short = get_top_n(-1*prev_returns, top_bottom_n)
         project_helper.print_top(df_long, 'Longed Stocks')
         project_helper.print_top(df_short, 'Shorted Stocks')
```

```
ticker       A  AAL  AAP  AAPL  ABBV  ABC  ABT  ACN  ADBE  ADI ...   XL  XLNX  \
date                                                               ...
```

```
2013-07-31 nan  nan  nan   nan   nan  nan  nan  nan   nan  nan ...  nan   nan
2013-08-31 nan  nan  nan   nan   nan  nan  nan  nan   nan  nan ...  nan   nan

ticker      XOM  XRAY  XRX  XYL  YUM  ZBH  ZION  ZTS
date
2013-07-31  nan   nan  nan  nan  nan  nan   nan  nan
2013-08-31  nan   nan  nan  nan  nan  nan   nan  nan

[2 rows x 495 columns]
ticker        A  AAL  AAP  AAPL  ABBV  ABC  ABT  ACN  ADBE  ADI ...    XL  XLNX  \
date                                                           ...
2013-07-31 nan  nan  nan   nan   nan  nan  nan  nan   nan  nan ...  nan   nan
2013-08-31 nan  nan  nan   nan   nan  nan  nan  nan   nan  nan ...  nan   nan

ticker      XOM  XRAY  XRX  XYL  YUM  ZBH  ZION  ZTS
date
2013-07-31  nan   nan  nan  nan  nan  nan   nan  nan
2013-08-31  nan   nan  nan  nan  nan  nan   nan  nan

[2 rows x 495 columns]
10 Most Longed Stocks:
INCY, AMD, AVGO, NFX, SWKS, NFLX, ILMN, UAL, NVDA, MU
10 Most Shorted Stocks:
RRC, FCX, CHK, MRO, GPS, WYNN, DVN, FTI, SPLS, TRIP
```

1.8 Projected Returns

It's now time to check if your trading signal has the potential to become profitable!

We'll start by computing the net returns this portfolio would return. For simplicity, we'll assume every stock gets an equal dollar amount of investment. This makes it easier to compute a portfolio's returns as the simple arithmetic average of the individual stock returns.

Implement the `portfolio_returns` function to compute the expected portfolio returns. Using `df_long` to indicate which stocks to long and `df_short` to indicate which stocks to short, calculate the returns using `lookahead_returns`. To help with calculation, we've provided you with `n_stocks` as the number of stocks we're investing in a single period.

```
In [14]: def portfolio_returns(df_long, df_short, lookahead_returns, n_stocks):
             """
             Compute expected returns for the portfolio, assuming equal investment in each long/

             Parameters
             ----------
             df_long : DataFrame
                 Top stocks for each ticker and date marked with a 1
             df_short : DataFrame
                 Bottom stocks for each ticker and date marked with a 1
             lookahead_returns : DataFrame
```

```
            Lookahead returns for each ticker and date
        n_stocks: int
            The number number of stocks chosen for each month

        Returns
        -------
        portfolio_returns : DataFrame
            Expected portfolio returns for each ticker and date
        """
        # TODO: Implement Function
        expected_returns = (df_long*lookahead_returns-df_short*lookahead_returns)/n_stocks
        return expected_returns



    project_tests.test_portfolio_returns(portfolio_returns)
```

```
Tests Passed
```

1.8.1 View Data

Time to see how the portfolio did.

```
In [15]: expected_portfolio_returns = portfolio_returns(df_long, df_short, lookahead_returns, 2*
         project_helper.plot_returns(expected_portfolio_returns.T.sum(), 'Portfolio Returns')
```

1.9 Statistical Tests

1.9.1 Annualized Rate of Return

```
In [16]: expected_portfolio_returns_by_date = expected_portfolio_returns.T.sum().dropna()
         portfolio_ret_mean = expected_portfolio_returns_by_date.mean()
         portfolio_ret_ste = expected_portfolio_returns_by_date.sem()
         portfolio_ret_annual_rate = (np.exp(portfolio_ret_mean * 12) - 1) * 100

         print("""
         Mean:                       {:.6f}
         Standard Error:             {:.6f}
         Annualized Rate of Return:  {:.2f}%
         """.format(portfolio_ret_mean, portfolio_ret_ste, portfolio_ret_annual_rate))
```

```
Mean:                       0.003185
Standard Error:             0.002158
Annualized Rate of Return:  3.90%
```

The annualized rate of return allows you to compare the rate of return from this strategy to other quoted rates of return, which are usually quoted on an annual basis.

1.9.2 T-Test

Our null hypothesis (H_0) is that the actual mean return from the signal is zero. We'll perform a one-sample, one-sided t-test on the observed mean return, to see if we can reject H_0.

We'll need to first compute the t-statistic, and then find its corresponding p-value. The p-value will indicate the probability of observing a t-statistic equally or more extreme than the one we observed if the null hypothesis were true. A small p-value means that the chance of observing the t-statistic we observed under the null hypothesis is small, and thus casts doubt on the null hypothesis. It's good practice to set a desired level of significance or alpha (α) *before* computing the p-value, and then reject the null hypothesis if $p < \alpha$.

For this project, we'll use $\alpha = 0.05$, since it's a common value to use.

Implement the `analyze_alpha` function to perform a t-test on the sample of portfolio returns. We've imported the `scipy.stats` module for you to perform the t-test.

Note: `scipy.stats.ttest_1samp` performs a two-sided test, so divide the p-value by 2 to get 1-sided p-value

```
In [17]: from scipy import stats


         def analyze_alpha(expected_portfolio_returns_by_date):
             """
             Perform a t-test with the null hypothesis being that the expected mean return is ze

             Parameters
             ----------
             expected_portfolio_returns_by_date : Pandas Series
                 Expected portfolio returns for each date

             Returns
             -------
             t_value
                 T-statistic from t-test
             p_value
                 Corresponding p-value
             """
             # TODO: Implement Function

             t,p = stats.ttest_1samp(expected_portfolio_returns_by_date,0)
             return t,p/2

         project_tests.test_analyze_alpha(analyze_alpha)
```

```
Tests Passed
```

1.9.3 View Data

Let's see what values we get with our portfolio. After you run this, make sure to answer the question below.

```
In [18]: t_value, p_value = analyze_alpha(expected_portfolio_returns_by_date)
         print("""
         Alpha analysis:
          t-value:        {:.3f}
          p-value:        {:.6f}
         """.format(t_value, p_value))
```

```
Alpha analysis:
 t-value:        1.476
 p-value:        0.073339
```

1.9.4 Question: What p-value did you observe? And what does that indicate about your signal?

Since pvalue > 0.05, hence we fail to reject the null hypothesis. This leads us to conclude that our strategy didn't contain an alpha. Cell*

1.10 Submission

Now that you're done with the project, it's time to submit it. Click the submit button in the bottom right. One of our reviewers will give you feedback on your project with a pass or not passed grade. You can continue to the next section while you wait for feedback.