project_1_starter

October 15, 2018

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 [1]: 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/a1/59/2613468ffbbe3a818934d06b81b9f4877fe
    100% || 880kB 686kB/s ta 0:00:011   24% |                          | 215kB 4.8MB/s eta 0:00:0
Requirement already satisfied: cycler==0.10.0 in /opt/conda/lib/python3.6/site-packages/cycler-
Collecting numpy==1.13.3 (from -r requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/57/a7/e3e6bd9d595125e1abbe162e323fd2d06f6
    100% || 17.0MB 39kB/s  eta 0:00:01  1% |                                | 296kB 9.2MB/s eta
Collecting pandas==0.21.1 (from -r requirements.txt (line 5))
  Downloading https://files.pythonhosted.org/packages/3a/e1/6c514df670b887c77838ab856f57783c07e
    100% || 26.2MB 24kB/s  eta 0:00:01  7% |                              | 2.1MB 20.4MB/s eta 0
Collecting plotly==2.2.3 (from -r requirements.txt (line 6))
```

```
  Downloading https://files.pythonhosted.org/packages/99/a6/8214b6564bf4ace9bec8a26e7f89832792
    100% || 1.1MB 600kB/s ta 0:00:011
Requirement already satisfied: pyparsing==2.2.0 in /opt/conda/lib/python3.6/site-packages (fro
Requirement already satisfied: python-dateutil==2.6.1 in /opt/conda/lib/python3.6/site-package
Requirement already satisfied: pytz==2017.3 in /opt/conda/lib/python3.6/site-packages (from -r
Requirement already satisfied: requests==2.18.4 in /opt/conda/lib/python3.6/site-packages (fro
Collecting scipy==1.0.0 (from -r requirements.txt (line 11))
  Downloading https://files.pythonhosted.org/packages/d8/5e/caa01ba7be11600b6a9d39265440d7b3be
    100% || 50.0MB 12kB/s  eta 0:00:01  0% |                                | 481kB 16.7MB/s eta
Requirement already satisfied: scikit-learn==0.19.1 in /opt/conda/lib/python3.6/site-packages
Requirement already satisfied: six==1.11.0 in /opt/conda/lib/python3.6/site-packages (from -r r
Collecting tqdm==4.19.5 (from -r requirements.txt (line 14))
  Downloading https://files.pythonhosted.org/packages/71/3c/341b4fa23cb3abc335207dba057c790f3b
    100% || 61kB 3.7MB/s eta 0:00:01
Collecting osqp (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/43/f2/bbeb83c0da6fd89a6d835b98d85ec76c04
    100% || 153kB 3.5MB/s eta 0:00:01
Collecting ecos>=2 (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/b6/b4/988b15513b13e8ea2eac65e97d84221ac5
    100% || 122kB 4.5MB/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/b3/fd/6e01c4f4a69fcc6c3db130ba55572089e7
    100% || 143kB 3.5MB/s eta 0:00:01
Collecting multiprocess (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/7a/ee/b9bf3e171f936743758ef924622d8dd005
    100% || 1.4MB 476kB/s eta 0:00:01  5% |                                | 81kB 12.0MB/s eta 0
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 (fro
Requirement already satisfied: nbformat>=4.2 in /opt/conda/lib/python3.6/site-packages (from pl
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 r
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 (fr
Requirement already satisfied: future in /opt/conda/lib/python3.6/site-packages (from osqp->cvx
Collecting dill>=0.2.8.1 (from multiprocess->cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/6f/78/8b96476f4ae426db71c6e86a8e6a81407f
    100% || 153kB 3.3MB/s ta 0:00:01
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.6/site-packages (fro
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /opt/conda/lib/python3.6/site-packag
Requirement already satisfied: traitlets>=4.1 in /opt/conda/lib/python3.6/site-packages (from n
Requirement already satisfied: jupyter-core in /opt/conda/lib/python3.6/site-packages (from nbf
Building wheels for collected packages: cvxpy, plotly, ecos, scs, multiprocess, dill
  Running setup.py bdist_wheel for cvxpy ... done
  Stored in directory: /root/.cache/pip/wheels/2b/60/0b/0c2596528665e21d698d6f84a3406c52044c7b4
  Running setup.py bdist_wheel for plotly ... done
  Stored in directory: /root/.cache/pip/wheels/98/54/81/dd92d5b0858fac680cd7bdb8800eb26c001dd9f
  Running setup.py bdist_wheel for ecos ... done
```

```
  Stored in directory: /root/.cache/pip/wheels/50/91/1b/568de3c087b3399b03d130e71b1fd048ec072c
  Running setup.py bdist_wheel for scs ... done
  Stored in directory: /root/.cache/pip/wheels/ff/f0/aa/530ccd478d7d9900b4e9ef5bc5a39e895ce110
  Running setup.py bdist_wheel for multiprocess ... done
  Stored in directory: /root/.cache/pip/wheels/8b/36/e5/96614ab62baf927e9bc06889ea794a8e87552b
  Running setup.py bdist_wheel for dill ... done
  Stored in directory: /root/.cache/pip/wheels/e2/5d/17/f87cb7751896ac629b435a8696f83ee75b1102
Successfully built cvxpy plotly ecos scs multiprocess dill
Installing collected packages: numpy, scipy, osqp, ecos, scs, dill, multiprocess, cvxpy, panda
  Found existing installation: numpy 1.12.1
    Uninstalling numpy-1.12.1:
      Successfully uninstalled numpy-1.12.1
  Found existing installation: scipy 0.19.1
    Uninstalling scipy-0.19.1:
      Successfully uninstalled scipy-0.19.1
  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 0.20.3
    Uninstalling pandas-0.20.3:
      Successfully uninstalled pandas-0.20.3
  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-0.2.8.2 ecos-2.0.5 multiprocess-0.70.6.1 numpy-1.13.3
You are using pip version 9.0.1, however version 18.1 is available.You should consider upgradi
```

1.2.2 Load Packages

```
In [2]: 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'], inde
```

```
        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]: project_helper.print_dataframe(close)
```

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/timese

            Returns
            -------
            prices_resampled : DataFrame
                Resampled prices for each ticker and date
            """
            # TODO: Implement Function
```

```
            # Methods to use (i.e. .first(), .max(), .min(), .close())
            # can be found here
            # https://pandas.pydata.org/pandas-docs/version/0.21.0/api.html#id44
            prices_resampled = close_prices.resample(freq).last()


            return prices_resampled

        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
```

```
        log_returns = np.log(prices) - np.log(prices.shift(1))

        return log_returns

    project_tests.test_compute_log_returns(compute_log_returns)
```

```
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))
```

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
             shifted_returns = returns.shift(shift_n)

             return shifted_returns

         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 [19]: # DEBUG
         #
         prev_returns
```

Out[19]:

ticker	A	AAL	AAP	AAPL	ABBV	\
date						
2013-07-31	nan	nan	nan	nan	nan	
2013-08-31	nan	nan	nan	nan	nan	
2013-09-30	0.04181412	-0.18015337	-0.02977582	0.08044762	-0.06518370	
2013-10-31	0.09657861	0.15979244	0.03282284	-0.02171531	0.04855545	
2013-11-30	-0.00960698	0.14734639	0.18195865	0.09201927	0.08860637	
2013-12-31	0.05388057	0.06647111	0.01828314	0.06772063	0.00000000	
2014-01-31	0.06769609	0.07267716	0.09197258	0.00886237	0.08616823	
2014-02-28	0.01664682	0.28421071	0.03663543	-0.11394918	-0.06220213	
2014-03-31	-0.02120344	0.09598737	0.10373913	0.05588347	0.03355617	
2014-04-30	-0.01790035	-0.00897599	-0.00629368	0.01975642	0.00957880	
2014-05-31	-0.03182502	-0.04270218	-0.04205794	0.09476126	0.02214224	
2014-06-30	0.05227357	0.13552541	0.02346722	0.07577509	0.04229556	

```
2014-07-31  0.01103586  0.06739797  0.08351026  0.02728625  0.03810157
2014-08-31 -0.02378338 -0.09799068 -0.10798279  0.02832630 -0.06780436
2014-09-30  0.01889681  0.00079762  0.11903914  0.07465224  0.05465028
2014-10-31 -0.00082047 -0.09153654 -0.04545245 -0.01722060  0.04388532
2014-11-30 -0.03028915  0.15558158  0.12032096  0.06948903  0.10172478
2014-12-31  0.06740128  0.16010974  0.00081622  0.10071836  0.08659108
2015-01-31 -0.04302767  0.09992645  0.08005917 -0.07460613 -0.05586717
2015-02-28 -0.07812999 -0.08865699 -0.00175946  0.05961156 -0.07327254
2015-03-31  0.11114202 -0.02228945 -0.02593080  0.09598802  0.00248242
2015-04-30 -0.01333009  0.09701683 -0.03400439 -0.03187426 -0.03293308
2015-05-31 -0.00434154 -0.08694248 -0.04572186  0.00576971  0.10773129
2015-06-30 -0.00436047 -0.13068039  0.06903017  0.04431573  0.02941164
2015-07-31 -0.06296587 -0.05918743  0.03922895 -0.03797801  0.00897001
2015-08-31  0.05961780  0.00412320  0.08953503 -0.03344115  0.04841767
2015-09-30 -0.12025862 -0.02595965  0.00595206 -0.06849869 -0.11491257
2015-10-31 -0.05317503 -0.00385555  0.07866971 -0.02523595 -0.13717756
2015-11-30  0.09523073  0.17421946  0.04588907  0.08329065  0.09970324
2015-12-31  0.10212435 -0.11137087 -0.19834401 -0.00580146 -0.02379042
2016-01-31  0.00250761  0.02607491 -0.07765707 -0.11679029  0.01857111
2016-02-29 -0.10480259 -0.08266322  0.01017989 -0.07822348 -0.06500804
2016-03-31 -0.00800004  0.05307585 -0.02402858 -0.00133143 -0.00529633
2016-04-30  0.06478949  0.00024387  0.07749311  0.11974615  0.04493730
2016-05-31  0.02934536 -0.16736448 -0.02679973 -0.15073114  0.07532447
2016-06-30  0.11462829 -0.08063179 -0.01458373  0.06933886  0.03114913
2016-07-31 -0.03131999 -0.11970434  0.04978712 -0.04359639 -0.01634128
2016-08-31  0.08115189  0.22631760  0.04966480  0.08623527  0.07621369
2016-09-30 -0.02376808  0.02526745 -0.07631131  0.02334347 -0.03268917
2016-10-31  0.00477784  0.00850369 -0.05350196  0.06344815 -0.01619914
2016-11-30 -0.07769182  0.10344664 -0.06253766  0.00432500 -0.11356073
2016-12-31  0.00936614  0.13692923  0.19193633 -0.02178228  0.08617441
2017-01-31  0.03818318  0.00536885 -0.00319576  0.04684076  0.02949493
2017-02-28  0.07217773 -0.05367464 -0.02927975  0.04664167 -0.01399583
2017-03-31  0.04648289  0.04873915 -0.04757013  0.12552425  0.01187489
2017-04-30  0.03263890 -0.09164993 -0.05439459  0.04754147  0.05229538
2017-05-31  0.04040579  0.00753654 -0.04215500 -0.00006961  0.02182202
2017-06-30  0.09175337  0.12956238 -0.06173770  0.06557216  0.00121249

ticker             ABC         ABT         ACN        ADBE         ADI  \
date
2013-07-31         nan         nan         nan         nan         nan
2013-08-31         nan         nan         nan         nan         nan
2013-09-30 -0.01609335 -0.09440968 -0.02136190 -0.03289558 -0.05749847
2013-10-31  0.07086509 -0.00420927  0.01905603  0.12689741  0.01650096
2013-11-30  0.06693948  0.10058564  0.01124304  0.04296064  0.04671322
2013-12-31  0.07997603  0.04389252  0.05260536  0.04613431 -0.02215021
2014-01-31 -0.00312412  0.00365918  0.05950782  0.05314171  0.06162558
2014-02-28 -0.04494321 -0.03893724 -0.02887307 -0.01157325 -0.05364189
2014-03-31  0.01278407  0.08167803  0.04252310  0.14797715  0.05873111
```

```
2014-04-30 -0.03387614 -0.03244633 -0.04452811 -0.04302219  0.04463996
2014-05-31 -0.00627057  0.01187986  0.01803285 -0.06358573 -0.03543414
2014-06-30  0.11924492  0.03225676  0.01521647  0.04516334  0.02805861
2014-07-31 -0.00713113  0.02200064 -0.00751744  0.11436847  0.03175466
2014-08-31  0.05684488  0.03470536 -0.01948600 -0.04393196 -0.08568401
2014-09-30  0.00928406  0.00284495  0.02220373  0.03755456  0.02958032
2014-10-31 -0.00116362 -0.01550789  0.00320237 -0.03841992 -0.02506897
2014-11-30  0.09976923  0.05225014  0.01089553  0.01335172  0.00262335
2014-12-31  0.06719583  0.02088612  0.06224762  0.04952333  0.09637224
2015-01-31 -0.00982294  0.01139295  0.03393630 -0.01339001  0.02250327
2015-02-28  0.05281720 -0.00042641 -0.06093959 -0.03599072 -0.06348240
2015-03-31  0.08089383  0.05667423  0.06896907  0.12036221  0.12320729
2015-04-30  0.10090334 -0.02219942  0.03984851 -0.06745892  0.07342445
2015-05-31  0.00552706  0.00708541 -0.00049630  0.02826855 -0.01858432
2015-06-30 -0.01281413  0.04589313  0.03593582  0.03906394  0.10020918
2015-07-31 -0.05684490  0.00982809  0.00767559  0.02398616 -0.05714976
2015-08-31 -0.00556369  0.03709245  0.06335602  0.01202468 -0.09565300
2015-09-30 -0.05271992 -0.11264239 -0.08963337 -0.04260734 -0.03591771
2015-10-31 -0.05179848 -0.11872187  0.04145402  0.04540863  0.00979789
2015-11-30  0.01587501  0.11369172  0.09776101  0.07541025  0.06369612
2015-12-31  0.02533250  0.00267499  0.00018655  0.03109289  0.02480620
2016-01-31  0.05012168 -0.00022264 -0.02569573  0.02675456 -0.10145040
2016-02-29 -0.14668975 -0.16459543  0.00990296 -0.05256030 -0.02674642
2016-03-31 -0.02946371  0.02324168 -0.05132322 -0.04568157 -0.00821101
2016-04-30 -0.00080845  0.07674113  0.14063754  0.09675045  0.11064939
2016-05-31 -0.01689525 -0.06663976 -0.01220169  0.00446762 -0.04970290
2016-06-30 -0.12208554  0.01859216  0.05219176  0.05422362  0.04515902
2016-07-31  0.05626889 -0.00810747 -0.04892046 -0.03769780 -0.03231130
2016-08-31  0.07139375  0.13578830 -0.00424592  0.02137959  0.11950785
2016-09-30  0.02449340 -0.06294575  0.01922714  0.04446953 -0.01342097
2016-10-31 -0.07383382  0.00640496  0.06048139  0.05911135  0.02976363
2016-11-30 -0.13867316 -0.06852950 -0.03945196 -0.00953490 -0.00544537
2016-12-31  0.10806742 -0.03027006  0.02707341 -0.04470124  0.15250011
2017-01-31  0.00256115  0.00889127 -0.01944600  0.00136081 -0.02206272
2017-02-28  0.10997958  0.09029065 -0.02822698  0.09650158  0.03144743
2017-03-31  0.05135996  0.07626031  0.07305358  0.04281683  0.09464624
2017-04-30 -0.03344570 -0.01497402 -0.02161979  0.09497211  0.00024408
2017-05-31 -0.07567373 -0.01142815  0.02214547  0.02736367 -0.07274784
2017-06-30  0.11615820  0.04524848  0.02579791  0.05894304  0.11819123

ticker         ...              XL        XLNX         XOM        XRAY  \
date           ...
2013-07-31     ...             nan         nan         nan         nan
2013-08-31     ...             nan         nan         nan         nan
2013-09-30     ...     -0.05879217 -0.06720501 -0.06596571 -0.02097402
2013-10-31     ...      0.04630944  0.07605219 -0.01293321  0.03481157
2013-11-30     ...     -0.00814469 -0.03082169  0.04076620  0.08157415
2013-12-31     ...      0.04540422 -0.01674184  0.04899644  0.00961292
```

```
2014-01-31     ...     -0.00005037  0.03298584  0.07935125  0.02044076
2014-02-28     ...     -0.10243348  0.01082965 -0.09357256 -0.04947109
2014-03-31     ...      0.05615273  0.12286783  0.05062164 -0.01660877
2014-04-30     ...      0.03284340  0.03888909  0.01454009  0.01589505
2014-05-31     ...      0.00319489 -0.13978958  0.04728753 -0.03110431
2014-06-30     ...      0.03479430  0.00123427 -0.01178295  0.05789258
2014-07-31     ...      0.01315416  0.00742551  0.00149098  0.00264665
2014-08-31     ...     -0.01508417 -0.13998391 -0.01743356 -0.01983642
2014-09-30     ...      0.05843276  0.03386045  0.01220640  0.02730581
2014-10-31     ...     -0.02525829  0.00236407 -0.05592900 -0.04369616
2014-11-30     ...      0.02117902  0.04253010  0.02789025  0.10739791
2014-12-31     ...      0.04727105  0.03443096 -0.05880687  0.07966385
2015-01-31     ...     -0.02838693 -0.04847111  0.02087622 -0.03037006
2015-02-28     ...      0.00348534 -0.11531726 -0.05594387 -0.06285044
2015-03-31     ...      0.04838969  0.10127687  0.02024303  0.05795769
2015-04-30     ...      0.02092430 -0.00165348 -0.04080317 -0.03938489
2015-05-31     ...      0.00757990  0.02475027  0.02750072  0.00215919
2015-06-30     ...      0.01605171  0.09654661 -0.01677365  0.02009094
2015-07-31     ...     -0.00860127 -0.07122468 -0.02375409 -0.00797670
2015-08-31     ...      0.02180358 -0.05611995 -0.04914479  0.09891886
2015-09-30     ...     -0.01938714  0.01075955 -0.04204308 -0.08237366
2015-10-31     ...     -0.02096434  0.01139343 -0.01189933 -0.03437113
2015-11-30     ...      0.04732066  0.11681228  0.10691949  0.18505330
2015-12-31     ...      0.00262261  0.04908477 -0.00427045 -0.00312732
2016-01-31     ...      0.03118510 -0.05629461 -0.04649669  0.00432337
2016-02-29     ...     -0.07745121  0.06789938 -0.00128370 -0.03274052
2016-03-31     ...     -0.05324021 -0.05604739  0.03819057  0.03454661
2016-04-30     ...      0.07359389  0.00443741  0.04202402  0.01220241
2016-05-31     ...     -0.11720576 -0.09619609  0.05594807 -0.03349319
2016-06-30     ...      0.04830993  0.10290823  0.01539863  0.04205613
2016-07-31     ...     -0.02466162 -0.02694774  0.05166393 -0.00071206
2016-08-31     ...      0.03828501  0.10192953 -0.05245058  0.03172855
2016-09-30     ...     -0.01104021  0.06579968 -0.01191629 -0.04112133
2016-10-31     ...     -0.01178072  0.00239521  0.00160532 -0.03228879
2016-11-30     ...      0.03132116 -0.06598725 -0.04643273 -0.03179755
2016-12-31     ...      0.04038386  0.06594093  0.05541489  0.01054005
2017-01-31     ...      0.03615589  0.11187869  0.03334393 -0.00643281
2017-02-28     ...      0.00828549 -0.03660694 -0.07318798 -0.01800276
2017-03-31     ...      0.07484917  0.01624776 -0.02195182  0.11358061
2017-04-30     ...     -0.01017620 -0.01593728  0.00844920 -0.01575947
2017-05-31     ...      0.04871848  0.08633458 -0.00439937  0.01273092
2017-06-30     ...      0.04302745  0.06090411 -0.00482798  0.00441780

ticker             XRX         XYL         YUM         ZBH        ZION  \
date
2013-07-31         nan         nan         nan         nan         nan
2013-08-31         nan         nan         nan         nan         nan
2013-09-30  0.02845720 -0.00134868 -0.04058203 -0.05402072 -0.05663911
```

```
2013-10-31  0.03611367  0.11966450  0.01937689  0.04028369 -0.01985983
2013-11-30 -0.03460553  0.21535825 -0.04870385  0.06287079  0.03494040
2013-12-31  0.13529041  0.00173762  0.13869403  0.04405904  0.03381174
2014-01-31  0.07181852  0.00115674 -0.02700927  0.02143144  0.02125228
2014-02-28 -0.11480883 -0.03649610 -0.11361565  0.00833515 -0.04122539
2014-03-31  0.01282069  0.16849675  0.09821667 -0.00138443  0.08307315
2014-04-30  0.03352930 -0.07737790  0.01752905  0.01022814 -0.00707626
2014-05-31  0.06757594  0.03161998  0.02580642  0.02320097 -0.06880844
2014-06-30  0.02127740 -0.00437311  0.00414777  0.07510359 -0.01005896
2014-07-31  0.01231307  0.04661750  0.04909193 -0.00262106  0.03031579
2014-08-31  0.06383490 -0.10201119 -0.15259272 -0.03717391 -0.02230321
2014-09-30  0.04064098  0.05432073  0.04273371 -0.00762369  0.01245116
2014-10-31 -0.03818588 -0.04527723 -0.00623228  0.01452958 -0.00274914
2014-11-30  0.00377217  0.02786669  0.00399386  0.10103487 -0.00310185
2014-12-31  0.04993695  0.05302447  0.07274470  0.00939481 -0.02173021
2015-01-31 -0.00277876 -0.00706717 -0.05864438  0.01193901  0.01590983
2015-02-28 -0.05106548 -0.11012919 -0.00220105 -0.01170641 -0.17386913
2015-03-31  0.03579801  0.04975030  0.11525879  0.07134515  0.11109071
2015-04-30 -0.05485114 -0.01923136 -0.02990901 -0.02227720  0.00986330
2015-05-31 -0.11099678  0.05553898  0.09320743 -0.06758378  0.04826092
2015-06-30 -0.00698083 -0.00836304  0.04714907  0.03992801  0.02108818
2015-07-31 -0.06439073  0.01357979 -0.00033298 -0.04353182  0.09427086
2015-08-31  0.03509132 -0.07097951 -0.02148999 -0.04839200 -0.01732264
2015-09-30 -0.08026959 -0.05779847 -0.09545832 -0.00491259 -0.07084284
2015-10-31 -0.03699314  0.01225130  0.00225395 -0.09522347 -0.05165634
2015-11-30 -0.03556860  0.10289172 -0.11330487  0.10730688  0.04369368
2015-12-31  0.11648057  0.02852030  0.02231302 -0.03463718  0.04255223
2016-01-31  0.01407508 -0.02221712  0.00741965  0.01765935 -0.09297646
2016-02-29 -0.08641291 -0.01518318 -0.00271062 -0.03299779 -0.18540322
2016-03-31 -0.01446306  0.04398671  0.00138083 -0.02499238 -0.05907417
2016-04-30  0.15666230  0.08919201  0.12171562  0.09887961  0.12712017
2016-05-31 -0.15057286  0.02128769 -0.02279296  0.08224057  0.12814724
2016-06-30  0.03781749  0.07021508  0.03130475  0.05331681  0.02029299
2016-07-31 -0.04083362 -0.00022394  0.01006008 -0.01224782 -0.10885619
2016-08-31  0.08190528  0.06838052  0.08081880  0.08561205  0.10384723
2016-09-30 -0.04467244  0.06487578  0.01432323 -0.01173551  0.09555297
2016-10-31  0.03568895  0.03078357  0.00110181  0.00504351  0.01395900
2016-11-30 -0.03618485 -0.08180785 -0.04530849 -0.20992565  0.03764481
2016-12-31 -0.04394012  0.06826631  0.02387708 -0.03416331  0.21341820
2017-01-31 -0.06031217 -0.04075737 -0.00094697  0.01539556  0.07851429
2017-02-28  0.14387403 -0.00424973  0.03879000  0.13680848 -0.01994669
2017-03-31  0.07101104 -0.02065858 -0.00320978 -0.01061987  0.06402762
2017-04-30 -0.00497178  0.04271546 -0.02197891  0.04404029 -0.06676818
2017-05-31 -0.02064767  0.02341935  0.03320595 -0.02035146 -0.04804045
2017-06-30 -0.01683069  0.01750300  0.09965606 -0.00368417  0.00296435

ticker            ZTS
date
```

```
2013-07-31         nan
2013-08-31         nan
2013-09-30 -0.02238899
2013-10-31  0.06539579
2013-11-30  0.01922875
2013-12-31 -0.01623981
2014-01-31  0.04825498
2014-02-28 -0.07165581
2014-03-31  0.02150621
2014-04-30 -0.06940744
2014-05-31  0.04700792
2014-06-30  0.01443595
2014-07-31  0.04987535
2014-08-31  0.01963856
2014-09-30  0.07618433
2014-10-31  0.04172452
2014-11-30  0.00372782
2014-12-31  0.19371924
2015-01-31 -0.04320818
2015-02-28 -0.00509269
2015-03-31  0.07569476
2015-04-30  0.00432995
2015-05-31 -0.03944740
2015-06-30  0.11372257
2015-07-31 -0.03163852
2015-08-31  0.01563818
2015-09-30 -0.08592248
2015-10-31 -0.08581672
2015-11-30  0.04347995
2015-12-31  0.08408991
2016-01-31  0.02578879
2016-02-29 -0.10497356
2016-03-31 -0.04732782
2016-04-30  0.07662724
2016-05-31  0.06119388
2016-06-30  0.00825838
2016-07-31  0.00288761
2016-08-31  0.06149185
2016-09-30  0.01240540
2016-10-31  0.01765151
2016-11-30 -0.08441037
2016-12-31  0.05457719
2017-01-31  0.06064797
2017-02-28  0.02796748
2017-03-31 -0.03011775
2017-04-30  0.00112486
2017-05-31  0.05205758
2017-06-30  0.10432630
```

```
[48 rows x 495 columns]
```

In [72]:
```
# DEBUG
#
i = 0
stop_iter = 2
ds_keep = None
for index, row in prev_returns.iterrows():

    if i == stop_iter:
        ds_keep = row
        break

    # iterate
    i += 1
```

In [76]:
```
# DEBUG
#
ds_keep
```

Out[76]:
```
ticker
A       0.00000000
AAL     0.00000000
AAP     0.00000000
AAPL    0.00000000
ABBV    0.00000000
ABC     0.00000000
ABT     0.00000000
ACN     0.00000000
ADBE    0.00000000
ADI     0.00000000
ADM     0.00000000
ADP     0.00000000
ADS     0.00000000
ADSK    0.00000000
AEE     0.00000000
AEP     0.00000000
AES     0.00000000
AET     0.00000000
AFL     0.00000000
AGN     0.00000000
AIG     0.00000000
AIV     0.00000000
AIZ     0.00000000
AJG     0.00000000
AKAM    0.00000000
ALB     0.00000000
```

```
ALGN    0.00000000
ALK     0.00000000
ALL     0.00000000
ALLE    0.00000000
           ...
VRTX    0.00000000
VTR     0.00000000
VZ      0.00000000
WAT     0.00000000
WBA     0.00000000
WDC     0.00000000
WEC     0.00000000
WFC     0.00000000
WHR     0.00000000
WLTW    0.00000000
WM      0.00000000
WMB     0.00000000
WMT     0.00000000
WRK     0.00000000
WU      0.00000000
WY      0.00000000
WYN     0.00000000
WYNN    0.00000000
XEC     0.00000000
XEL     0.00000000
XL      0.00000000
XLNX    0.00000000
XOM     0.00000000
XRAY    0.00000000
XRX     0.00000000
XYL     0.00000000
YUM     0.00000000
ZBH     0.00000000
ZION    0.00000000
ZTS     0.00000000
Name: 2013-09-30 00:00:00, Length: 495, dtype: float64
```

In [75]:
```
# DEBUG
#
ds_keep[ds_keep.index] = 0
```

In [58]:
```
# DEBUG
#
ds_keep.nlargest(2)
```

Out[58]:
```
ticker
INCY    0.36995670
BBY     0.17932605
Name: 2013-09-30 00:00:00, dtype: float64
```

```
In [62]: # DEBUG
         #
         ds_keep.nlargest(2).index

Out[62]: Index(['INCY', 'BBY'], dtype='object', name='ticker')

In [60]: # DEBUG
         #
         ds_keep[ds_keep.nlargest(2).index]

Out[60]: ticker
         INCY   0.36995670
         BBY    0.17932605
         Name: 2013-09-30 00:00:00, dtype: float64

In [63]: # DEBUG
         #
         prev_returns_copy = prev_returns.copy()

In [65]: # DEBUG
         #
         prev_returns_copy[ds_keep.nlargest(2).index]

Out[65]: ticker            INCY         BBY
         date
         2013-07-31         nan         nan
         2013-08-31         nan         nan
         2013-09-30  0.36995670  0.17932605
         2013-10-31  0.11840577  0.04540360
         2013-11-30  0.02203589  0.13219717
         2013-12-31  0.17803890 -0.05400232
         2014-01-31  0.08294374 -0.01252810
         2014-02-28  0.25781115 -0.52717384
         2014-03-31 -0.01941809  0.12333730
         2014-04-30 -0.18288194 -0.00174837
         2014-05-31 -0.09725527 -0.01834213
         2014-06-30  0.02018211  0.06458673
         2014-07-31  0.13019587  0.12007052
         2014-08-31 -0.17097582 -0.04215310
         2014-09-30  0.13047860  0.07013584
         2014-10-31 -0.09984072  0.05789299
         2014-11-30  0.31274756  0.01624131
         2014-12-31  0.11920695  0.14354987
         2015-01-31 -0.03282953 -0.00577466
         2015-02-28  0.08642989 -0.10200261
         2015-03-31  0.07420654  0.07916820
         2015-04-30  0.06548449  0.00980934
         2015-05-31  0.05827303 -0.08674679
         2015-06-30  0.12548397  0.00144196
```

```
2015-07-31 -0.05543498 -0.05534500
2015-08-31  0.00067150 -0.00986141
2015-09-30  0.10814719  0.12910850
2015-10-31 -0.05129782  0.01632263
2015-11-30  0.06276466 -0.05795107
2015-12-31 -0.02839212 -0.09736768
2016-01-31 -0.05201226 -0.03532579
2016-02-29 -0.42982583 -0.08638461
2016-03-31  0.04082199  0.14814835
2016-04-30 -0.01411272  0.02424601
2016-05-31 -0.00276358 -0.01115945
2016-06-30  0.15527677  0.00280156
2016-07-31 -0.05390927 -0.04056841
2016-08-31  0.12036368  0.09352606
2016-09-30 -0.10645732  0.13561256
2016-10-31  0.15069218 -0.00026125
2016-11-30 -0.08081191  0.01893947
2016-12-31  0.16224868  0.16084701
2017-01-31 -0.01994537 -0.06265360
2017-02-28  0.18965803  0.04244243
2017-03-31  0.09357615 -0.00879870
2017-04-30  0.00427335  0.11517363
2017-05-31 -0.07283699  0.05270633
2017-06-30  0.03983019  0.13654268
```

In [68]:
```
# DEBUG
#
prev_returns_copy = 0
prev_returns_copy[ds_keep.nlargest(2).index] = 1
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-68-9621efdbce19> in <module>()
  2 #
  3 prev_returns_copy = 0
----> 4 prev_returns_copy[ds_keep.nlargest(2).index] = 1


TypeError: 'int' object does not support item assignment
```

In [14]:
```
# DEBUG
#
# list_temp_a = [1,2,3]
```

```
        # s = pd.Series(list_temp_a)

        # s.size

In [93]: 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



            # https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-data
            # https://knowledge.udacity.com/questions/10343
            #
            top_stocks = pd.DataFrame( dtype = np.int64 )

            # Cycle through each row.
            # According to the notes, a vecorization solution is too difficult.
            # I'll take their word for it.
            for index, row in prev_returns.iterrows():


                # Do not affect the actual row since it is mutable
                # in the iteration
                row_copy = row.copy()

                # Find the INDEX of the top n largest values.
                # NOTE: .nlargest() returns the n largest VALUES
                #       in the Series, in sorted order
                #
                #       by doing .nlargest().index, we get the indices for those values.
                #       We will use these indices as a mask.
                mask_indices = row_copy.nlargest(top_n).index

                # Zero-out all the values in the copy
```

```
                row_copy[row_copy.index] = 0

                # Make the indices of the top n largest values equal 1.
                row_copy[mask_indices] = 1

                # Cast to int64
                row_copy_int64 = row_copy.astype( np.int64,
                                                  copy=False )

                # Append to the result Pandas DataFrame
                # NOTE: .append() is NOT inplace
                # https://stackoverflow.com/questions/33094056/is-it-possible-to-append-series
                # http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.appe
                top_stocks = top_stocks.append( row_copy_int64,
                                                ignore_index=True )


            # Cast pandas DataFrame to dtype "int64"
            top_stocks = top_stocks.astype( np.int64 )

            # Make the index of the pandas DataFrame to be the same as
            # the prev_returns.index, which is a TimeStamp
            top_stocks.index = prev_returns.index

            # Return the modified pandas DataFrame
            return top_stocks

        project_tests.test_get_top_n(get_top_n)
```

```
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 [90]: 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')
```

```
10 Most Longed Stocks:
AMD, INCY, NFX, ILMN, AVGO, SWKS, NVDA, UAL, NFLX, MU
10 Most Shorted Stocks:
```

```
CHK, RRC, FCX, MRO, WYNN, DVN, FTI, AMD, KORS, NEM
```

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 [122]: def portfolio_returns(df_long, df_short, lookahead_returns, n_stocks):
              """
              Compute expected returns for the portfolio, assuming equal investment in each lo

              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
              """

              # NOTE: In the description for the function, the variable name for
              #       the return is the same as the function thus overriding it.
              #       I changed the variable name to result.
              result = lookahead_returns*( df_long - df_short ) / n_stocks

              return result

          project_tests.test_portfolio_returns(portfolio_returns)
```

```
Tests Passed
```

1.8.1 View Data

Time to see how the portfolio did.

```
In [124]: expected_portfolio_returns = portfolio_returns(df_long, df_short, lookahead_returns,
          project_helper.plot_returns(expected_portfolio_returns.T.sum(), 'Portfolio Returns')
```

1.9 Statistical Tests

1.9.1 Annualized Rate of Return

```
In [125]: 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:                         0.003303
Standard Error:               0.002201
Annualized Rate of Return:    4.04%
```

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 mean return 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 mean 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 [131]: 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

            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

            # Note: scipy.stats.ttest_1samp performs a two-sided test, so divide the p-value
            #
            # scipy.stats.ttest_1samp
            # https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_1samp.h
            t_value, p_value = stats.ttest_1samp( expected_portfolio_returns_by_date,
                                                  0 )
            p_value_half =  p_value/2

            return t_value, p_value_half

        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 [129]: 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:        1.500
 p-value:        0.070160
```

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

#*TODO: Put Answer In this Cell* The p-value I observed is: 0.070160

What this indicate is that we do not reject the null hypothesis because p-value is not less than the alpha, which is set at value 0.05.

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.