Project 1: Trading with Momentum

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.

Packages

When you implement the functions, you'll only need to you use the packages you've used in the classroom, like [Pandas (https://pandas.pydata.org/)](https://pandas.pydata.org/) and [Numpy (http://www.numpy.org/)](http://www.numpy.org/). 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.

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 requirements.
txt (line 1)) (0.1.5)
Collecting cvxpy==1.0.3 (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/a1/59/2613468ffbbe3a818934d06b81b9f4877fe054afbf4f99d2f
43f398a0b34/cvxpy-1.0.3.tar.gz (880kB)
    100% |████████████████████████████████| 880kB 7.2MB/s ta 0:00:01
Requirement already satisfied: cycler==0.10.0 in /opt/conda/lib/python3.6/site-packages/cycler-0.10.0-py3.6.e
gg (from -r requirements.txt (line 3)) (0.10.0)
Collecting numpy==1.13.3 (from -r requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/57/a7/e3e6bd9d595125e1abbe162e323fd2d06f6f6683185294b79
cd2cdb190d5/numpy-1.13.3-cp36-cp36m-manylinux1_x86_64.whl (17.0MB)
    100% |████████████████████████████████| 17.0MB 924kB/s eta 0:00:01  9% |███                             |
1.6MB 13.9MB/s eta 0:00:02    12% |████                            | 2.2MB 9.2MB/s eta 0:00:02    46% |██████
█████████                        | 7.9MB 15.0MB/s eta 0:00:01    50% |████████████████▏               | 8.6MB 13.1M
B/s eta 0:00:01    54% |█████████████████▌              | 9.2MB 14.1MB/s eta 0:00:01    57% |███████████████
███▍             | 9.8MB 16.1MB/s eta 0:00:01    61% |███████████████████▋            | 10.4MB 14.1MB/s eta
0:00:01    64% |████████████████████▋           | 11.0MB 10.1MB/s eta 0:00:01    67% |█████████████████████
▋          | 11.5MB 11.4MB/s eta 0:00:01    74% |███████████████████████▊        | 12.6MB 11.4MB/s eta 0:0
0:01    84% |███████████████████████████     | 14.3MB 14.1MB/s eta 0:00:01    87% |██████████████████████████
██    | 14.9MB 10.2MB/s eta 0:00:01    90% |█████████████████████████████   | 15.4MB 12.3MB/s eta 0:00:01
97% |███████████████████████████████▏| 16.6MB 12.4MB/s eta 0:00:01
Collecting pandas==0.21.1 (from -r requirements.txt (line 5))
  Downloading https://files.pythonhosted.org/packages/3a/e1/6c514df670b887c77838ab856f57783c07e8760f2e3d59392
03a39735e0e/pandas-0.21.1-cp36-cp36m-manylinux1_x86_64.whl (26.2MB)
    100% |████████████████████████████████| 26.2MB 805kB/s eta 0:00:01  0% |▏
| 215kB 14.8MB/s eta 0:00:02    3% |█                               | 890kB 19.9MB/s eta 0:00:02    5% |█▊
| 1.5MB 10.0MB/s eta 0:00:03    12% |████▏                           | 3.4MB 14.6MB/s eta 0:00:02    15% |██
███                          | 4.0MB 9.6MB/s eta 0:00:03    19% |██████▎                         | 5.1MB 1
4.9MB/s eta 0:00:02    27% |█████████                       | 7.3MB 13.1MB/s eta 0:00:02    37% |████████████
| 9.8MB 14.5MB/s eta 0:00:02    39% |████████████▋                   | 10.5MB 15.3MB/s eta 0:00:02    51% |█
████████████████                | 13.6MB 13.7MB/s eta 0:00:01    54% |█████████████████▍              | 14.2
MB 14.2MB/s eta 0:00:01    59% |███████████████████             | 15.5MB 12.3MB/s eta 0:00:01    61% |█████
██████████████▋             | 16.1MB 17.1MB/s eta 0:00:01    77% |█████████████████████████       | 20.5MB 13.3
MB/s eta 0:00:01    80% |█████████████████████████▋      | 21.0MB 11.6MB/s eta 0:00:01    84% |█████████████
██████████████     | 22.1MB 11.7MB/s eta 0:00:01    92% |█████████████████████████████▋  | 24.4MB 1.8MB/s et
a 0:00:02    95% |██████████████████████████████▌ | 25.0MB 15.3MB/s eta 0:00:01    97% |█████████████████████
██████████ | 25.5MB 11.4MB/s eta 0:00:01
Collecting plotly==2.2.3 (from -r requirements.txt (line 6))
  Downloading https://files.pythonhosted.org/packages/99/a6/8214b6564bf4ace9bec8a26e7f89832792be582c042c47c91
2d3201328a0/plotly-2.2.3.tar.gz (1.1MB)
    100% |████████████████████████████████| 1.1MB 15.8MB/s ta 0:00:01
Requirement already satisfied: pyparsing==2.2.0 in /opt/conda/lib/python3.6/site-packages (from -r requiremen
ts.txt (line 7)) (2.2.0)
```

```
Requirement already satisfied: python-dateutil==2.6.1 in /opt/conda/lib/python3.6/site-packages (from -r requ
irements.txt (line 8)) (2.6.1)
Requirement already satisfied: pytz==2017.3 in /opt/conda/lib/python3.6/site-packages (from -r requirements.t
xt (line 9)) (2017.3)
Requirement already satisfied: requests==2.18.4 in /opt/conda/lib/python3.6/site-packages (from -r requiremen
ts.txt (line 10)) (2.18.4)
Collecting scipy==1.0.0 (from -r requirements.txt (line 11))
  Downloading https://files.pythonhosted.org/packages/d8/5e/caa01ba7be11600b6a9d39265440d7b3be3d69206da887c42
bef049521f2/scipy-1.0.0-cp36-cp36m-manylinux1_x86_64.whl (50.0MB)
    100% |████████████████████████████████| 50.0MB 355kB/s ta 0:00:011  0% ||
| 225kB 21.2MB/s eta 0:00:03    5% |█▋                              | 2.7MB 25.1MB/s eta 0:00:02    7% |██▌
| 3.8MB 23.7MB/s eta 0:00:02    12% |████▏                           | 6.1MB 24.2MB/s eta 0:00:02    14% |█
███                             | 7.3MB 24.3MB/s eta 0:00:02    16% |█████▎                          | 8.5MB 
24.7MB/s eta 0:00:02    19% |██████▏                         | 9.6MB 26.2MB/s eta 0:00:02    23% |███████▍
| 11.5MB 19.1MB/s eta 0:00:03    25% |████████                        | 12.6MB 25.1MB/s eta 0:00:02    27% |█
████████                        | 13.8MB 19.5MB/s eta 0:00:02    29% |█████████▌                      | 14.9M
B 21.2MB/s eta 0:00:02    31% |██████████▏                     | 15.9MB 20.7MB/s eta 0:00:02    34% |███████
████                    | 17.1MB 17.9MB/s eta 0:00:02    42% |█████████████▋                  | 21.4MB 23.6
MB/s eta 0:00:02    47% |███████████████▏                | 23.7MB 21.2MB/s eta 0:00:02    52% |█████████████
███                 | 26.3MB 16.0MB/s eta 0:00:02    63% |████████████████████▏           | 31.5MB 19.9MB/s 
eta 0:00:01    64% |████████████████████▋           | 32.5MB 25.6MB/s eta 0:00:01    67% |██████████████████
███          | 33.6MB 22.1MB/s eta 0:00:01    69% |██████████████████████▏         | 34.7MB 28.1MB/s eta 0:
00:01    72% |███████████████████████▏        | 36.1MB 12.4MB/s eta 0:00:02    73% |███████████████████████▌
| 36.7MB 11.5MB/s eta 0:00:02    74% |███████████████████████▉        | 37.2MB 9.1MB/s eta 0:00:02    75% |█
███████████████████████▏        | 37.7MB 11.7MB/s eta 0:00:02    76% |████████████████████████▌       | 38.3M
B 12.6MB/s eta 0:00:01    77% |████████████████████████▉       | 38.7MB 9.5MB/s eta 0:00:02    79% |████████
█████████████████▍      | 39.6MB 10.7MB/s eta 0:00:01    83% |██████████████████████████▊     | 41.9MB 9.6M
B/s eta 0:00:01    84% |███████████████████████████▏    | 42.4MB 7.9MB/s eta 0:00:01    85% |███████████████
█████████████▌    | 42.9MB 10.2MB/s eta 0:00:01    87% |████████████████████████████    | 43.9MB 10.6MB/s eta
0:00:01    89% |████████████████████████████▋   | 44.8MB 10.0MB/s eta 0:00:01    92% |██████████████████████
███████▌  | 46.2MB 11.4MB/s eta 0:00:01    93% |██████████████████████████████  | 46.8MB 20.6MB/s eta 0:00:01
94% |██████████████████████████████▏ | 47.4MB 14.9MB/s eta 0:00:01    95% |██████████████████████████████▋
| 48.0MB 10.5MB/s eta 0:00:01    96% |███████████████████████████████ | 48.4MB 9.0MB/s eta 0:00:01    97% |██
█████████████████████████████▍| 49.0MB 13.6MB/s eta 0:00:01    99% |███████████████████████████████▊| 49.6M
B 9.3MB/s eta 0:00:01
Requirement already satisfied: scikit-learn==0.19.1 in /opt/conda/lib/python3.6/site-packages (from -r requir
ements.txt (line 12)) (0.19.1)
Requirement already satisfied: six==1.11.0 in /opt/conda/lib/python3.6/site-packages (from -r requirements.tx
t (line 13)) (1.11.0)
Collecting tqdm==4.19.5 (from -r requirements.txt (line 14))
  Downloading https://files.pythonhosted.org/packages/71/3c/341b4fa23cb3abc335207dba057c790f3bb329f6757e1fcd5
d347bcf8308/tqdm-4.19.5-py2.py3-none-any.whl (51kB)
    100% |████████████████████████████████| 61kB 10.5MB/s ta 0:00:01
```

```
Collecting osqp (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/6c/59/2b80e881be227eecef3f2b257339d182167b55d22a1315ff4
303ddcfd42f/osqp-0.6.1-cp36-cp36m-manylinux1_x86_64.whl (208kB)
    100% |████████████████████████████████| 215kB 8.3MB/s ta 0:00:011
Collecting ecos>=2 (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/55/ed/d131ff51f3a8f73420eb1191345eb49f269f23cadef515172
e356018cde3/ecos-2.0.7.post1-cp36-cp36m-manylinux1_x86_64.whl (147kB)
    100% |████████████████████████████████| 153kB 17.2MB/s ta 0:00:01
Collecting scs>=1.1.3 (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/f2/6e/dbdd778c64c1920ae357a2013ea655d65a1f8b60f397d6e55
49e4aafe8ec/scs-2.1.1-2.tar.gz (157kB)
    100% |████████████████████████████████| 163kB 15.5MB/s ta 0:00:01
Collecting multiprocess (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/58/17/5151b6ac2ac9b6276d46c33369ff814b0901872b2a0871771
252f02e9192/multiprocess-0.70.9.tar.gz (1.6MB)
    100% |████████████████████████████████| 1.6MB 11.7MB/s ta 0:00:01
Requirement already satisfied: fastcache in /opt/conda/lib/python3.6/site-packages (from cvxpy==1.0.3->-r req
uirements.txt (line 2)) (1.0.2)
Requirement already satisfied: toolz in /opt/conda/lib/python3.6/site-packages (from cvxpy==1.0.3->-r require
ments.txt (line 2)) (0.8.2)
Requirement already satisfied: decorator>=4.0.6 in /opt/conda/lib/python3.6/site-packages (from plotly==2.2.3
->-r requirements.txt (line 6)) (4.0.11)
Requirement already satisfied: nbformat>=4.2 in /opt/conda/lib/python3.6/site-packages (from plotly==2.2.3->-
r requirements.txt (line 6)) (4.4.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/python3.6/site-packages (from requests
==2.18.4->-r requirements.txt (line 10)) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in /opt/conda/lib/python3.6/site-packages (from requests==2.18.
4->-r requirements.txt (line 10)) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /opt/conda/lib/python3.6/site-packages (from requests
==2.18.4->-r requirements.txt (line 10)) (1.22)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.6/site-packages (from requests==
2.18.4->-r requirements.txt (line 10)) (2019.11.28)
Requirement already satisfied: future in /opt/conda/lib/python3.6/site-packages (from osqp->cvxpy==1.0.3->-r
requirements.txt (line 2)) (0.16.0)
Collecting dill>=0.3.1 (from multiprocess->cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/c7/11/345f3173809cea7f1a193bfbf02403fff250a3360e0e118a1
630985e547d/dill-0.3.1.1.tar.gz (151kB)
    100% |████████████████████████████████| 153kB 15.5MB/s ta 0:00:01
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.6/site-packages (from nbformat>=4.2
->plotly==2.2.3->-r requirements.txt (line 6)) (0.2.0)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /opt/conda/lib/python3.6/site-packages (from nbform
at>=4.2->plotly==2.2.3->-r requirements.txt (line 6)) (2.6.0)
Requirement already satisfied: traitlets>=4.1 in /opt/conda/lib/python3.6/site-packages (from nbformat>=4.2->
```

```
plotly==2.2.3->-r requirements.txt (line 6)) (4.3.2)
Requirement already satisfied: jupyter-core in /opt/conda/lib/python3.6/site-packages (from nbformat>=4.2->pl
otly==2.2.3->-r requirements.txt (line 6)) (4.4.0)
Building wheels for collected packages: cvxpy, plotly, scs, multiprocess, dill
  Running setup.py bdist_wheel for cvxpy ... done
  Stored in directory: /root/.cache/pip/wheels/2b/60/0b/0c2596528665e21d698d6f84a3406c52044c7b4ca6ac737cf3
  Running setup.py bdist_wheel for plotly ... done
  Stored in directory: /root/.cache/pip/wheels/98/54/81/dd92d5b0858fac680cd7bdb8800eb26c001dd9f5dc8b1bc0ba
  Running setup.py bdist_wheel for scs ... done
  Stored in directory: /root/.cache/pip/wheels/68/3f/24/e9c75d426f600634cdac68321184ba06fdc4ab2d189b5c4541
  Running setup.py bdist_wheel for multiprocess ... done
  Stored in directory: /root/.cache/pip/wheels/96/20/ac/9f1d164f7d81787cd6f4401b1d05212807d021fbbbcc301b82
  Running setup.py bdist_wheel for dill ... done
  Stored in directory: /root/.cache/pip/wheels/59/b1/91/f02e76c732915c4015ab4010f3015469866c1eb9b14058d8e7
Successfully built cvxpy plotly scs multiprocess dill
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, osqp, ecos, scs, dill, multiprocess, cvxpy, pandas, plotly, tqdm
  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-0.3.1.1 ecos-2.0.7.post1 multiprocess-0.70.9 numpy-1.13.3 osqp-0.6.1
 pandas-0.21.1 plotly-2.2.3 scipy-1.0.0 scs-2.1.1.post2 tqdm-4.19.5
```

Load Packages

In [29]:
```
import matplotlib as plt
```

In [24]:
```
%matplotlib inline
```

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

Market Data

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 [2]:
```
df = pd.read_csv('../../data/project_1/eod-quotemedia.csv', parse_dates=['date'], index_col=False)

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

print('Loaded Data')
```

```
Loaded Data
```

In [6]:
```
df.to_csv ('eod-quotemedia.csv')
```

View Data

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

In [8]:
```
close.shape
```

Out[8]:
```
(1009, 495)
```

In [20]:
```
close.head()
```

Out[20]:

ticker date	A	AAL	AAP	AAPL	ABBV	ABC	ABT	ACN	ADBE	
2013-07-01	29.99418563	16.17609308	81.13821681	53.10917319	34.92447839	50.86319750	31.42538772	64.69409505	46.23500000	39.9133
2013-07-02	29.65013670	15.81983388	80.72207258	54.31224742	35.42807578	50.69676639	31.27288084	64.71204071	46.03000000	39.8605
2013-07-03	29.70518453	16.12794994	81.23729877	54.61204262	35.44486235	50.93716689	30.72565028	65.21451912	46.42000000	40.1860
2013-07-05	30.43456826	16.21460758	81.82188233	54.17338125	35.85613355	51.37173702	31.32670680	66.07591068	47.00000000	40.6523
2013-07-08	30.52402098	16.31089385	82.95141667	53.86579916	36.66188936	52.03746147	31.76628544	66.82065546	46.62500000	40.2564

5 rows × 495 columns

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

	A	AAL	AAP	...
2013-07-0	29.994	16.176	81.138	...
2013-07-0	29.650	15.820	80.722	...
2013-07-0	29.705	16.128	81.237	...
2013-07-0	30.435	16.215	81.822	...
2013-07-0	30.524	16.311	82.951	...
2013-07-0	30.689	16.715	82.436	...
2013-07-1	31.178	16.532	81.990	...
2013-07-1	31.460	16.725	82.000	...
2013-07-1	31.480	16.908	81.911	...
2013-07-1	31.728	17.100	82.615	...
...	...	...	...	...

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 [32]:
```
close['AAPL'].plot(figsize=(16,8),color='r')
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6a6922b5f8>

In [21]:
```
apple_ticker = 'AAPL'
project_helper.plot_stock(close[apple_ticker], '{} Stock'.format(apple_ticker))
```

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 [45]:
```
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/timeseries.html#offset-aliase
s

    Returns
    -------
    prices_resampled : DataFrame
        Resampled prices for each ticker and date
    """
    # TODO: Implement Function
    monthly_close = close_prices.resample(freq).last()
    return monthly_close

project_tests.test_resample_prices(resample_prices)
```

```
Tests Passed
```

In [46]:
```
resample_prices(close).head()
```

Out[46]:

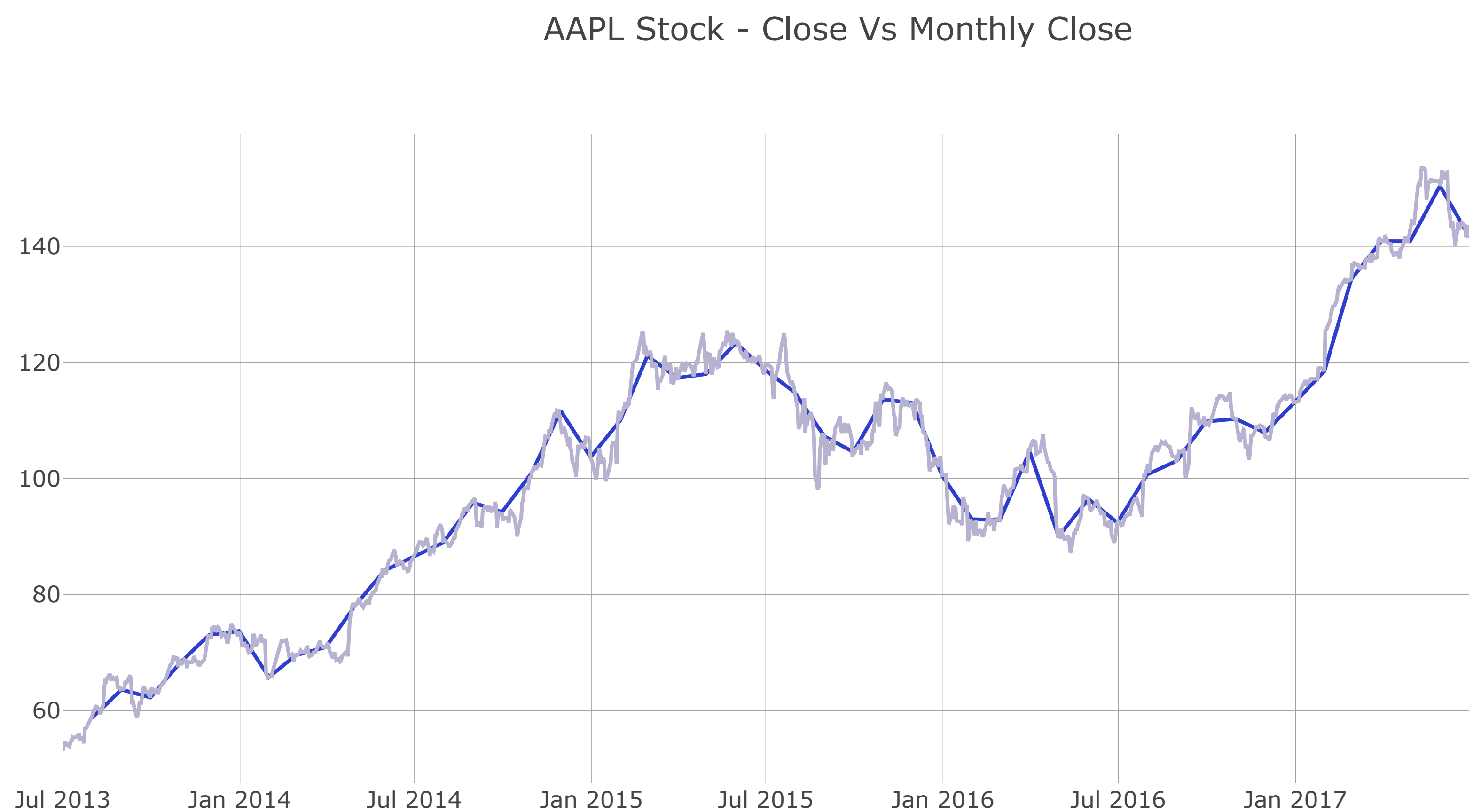
ticker	A	AAL	AAP	AAPL	ABBV	ABC	ABT	ACN	ADBE	
date										
2013-07-31	30.77861719	18.63139292	81.73270857	58.73000866	38.52144972	53.87744989	32.99081455	66.22844876	47.28000000	43.441
2013-08-31	32.09288410	15.55986096	79.33492514	63.64994327	36.09056668	53.01732111	30.01866909	64.82868748	45.75000000	41.013
2013-09-30	35.34697923	18.25587647	81.98212977	62.28266407	37.88620154	56.91072241	29.89257807	66.07591068	51.94000000	41.696
2013-10-31	35.00902763	21.15409315	98.34285959	68.28583759	41.39637606	60.85069550	33.05576095	66.82299697	54.22000000	43.690
2013-11-30	36.94707663	22.60801580	100.15741326	73.07037475	41.39637606	65.91719111	34.53897430	70.43234797	56.78000000	42.732

5 rows × 495 columns

View Data

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

In [47]:
```
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))
```



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](https://docs.scipy.org/doc/numpy/reference/generated/numpy.log.html) (https://docs.scipy.org/doc/numpy/reference/generated/numpy.log.html) to help you calculate the log returns.

In [56]:
```
s=pd.Series([1,1,2,3,5])
s-s.shift(1)
```

Out[56]:
```
0          nan
1   0.00000000
2   1.00000000
3   1.00000000
4   2.00000000
dtype: float64
```

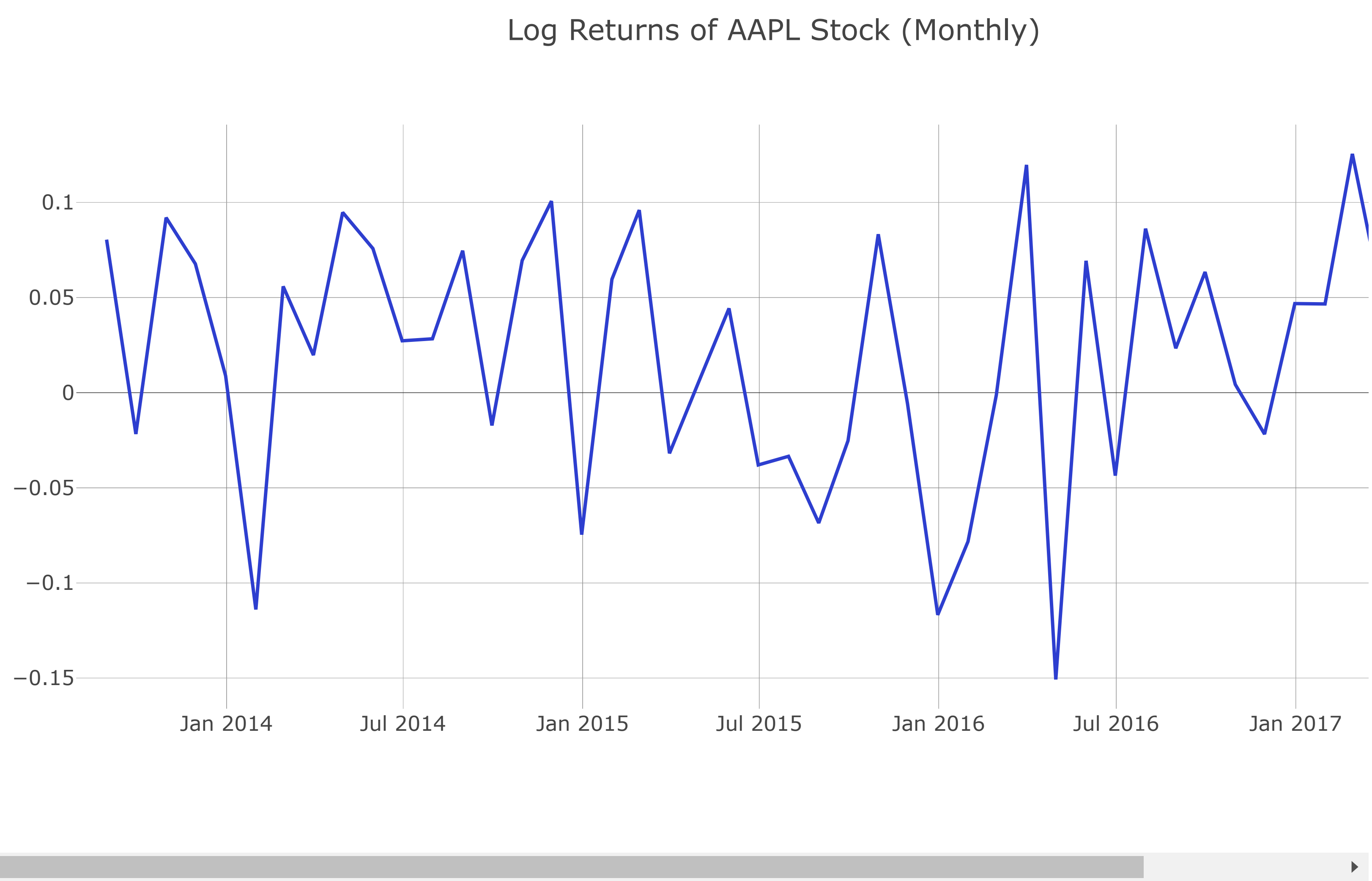
In [48]:
```
def compute_log_returns(prices):
    """
    Compute log returns for each ticker.

    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
```

View Data

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

In [57]:
```
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))
```



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.

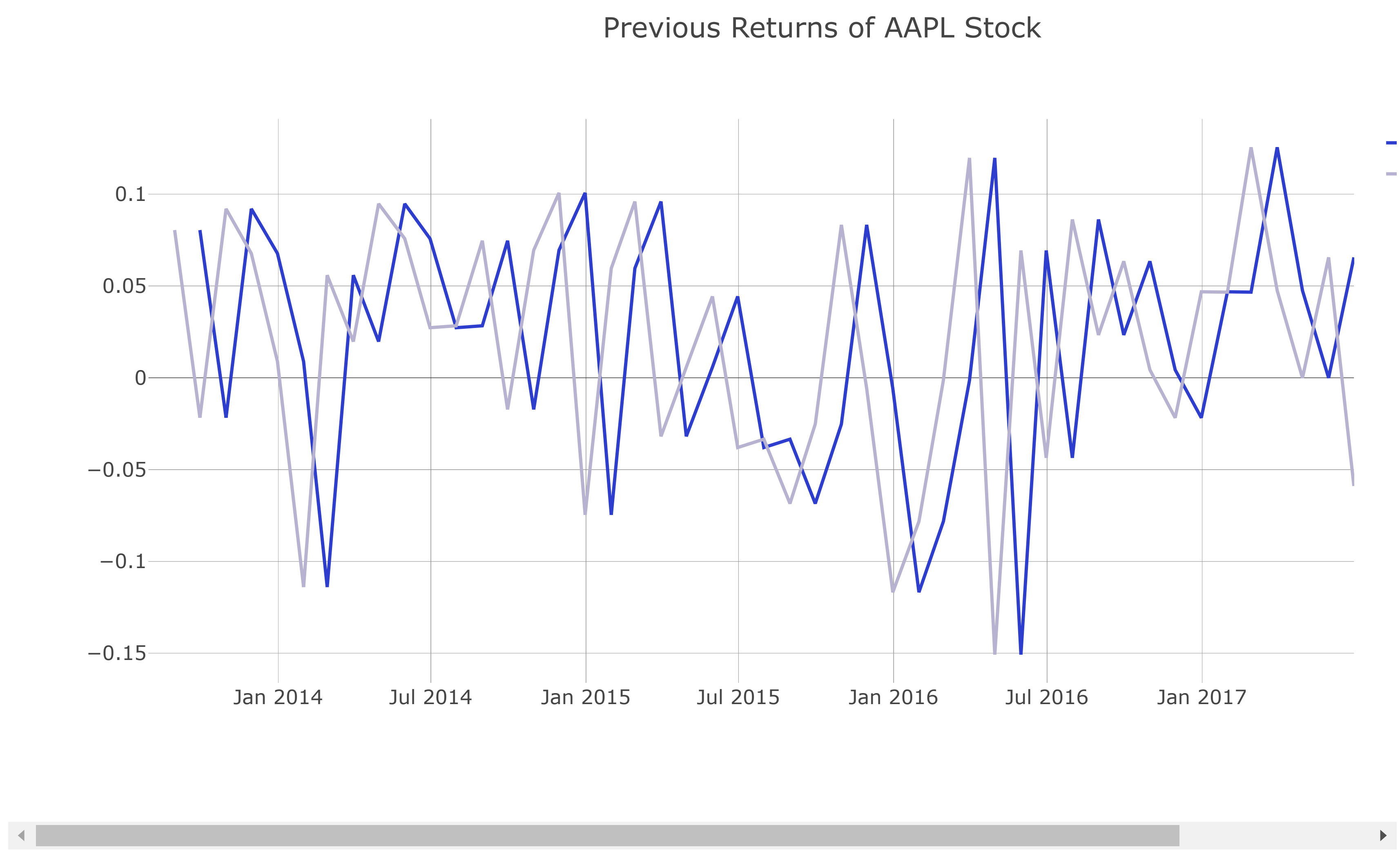
In [59]:
```
def shift_returns(returns, shift_n):
    """
    Generate shifted returns

    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
    return returns.shift(shift_n)

project_tests.test_shift_returns(shift_returns)
```

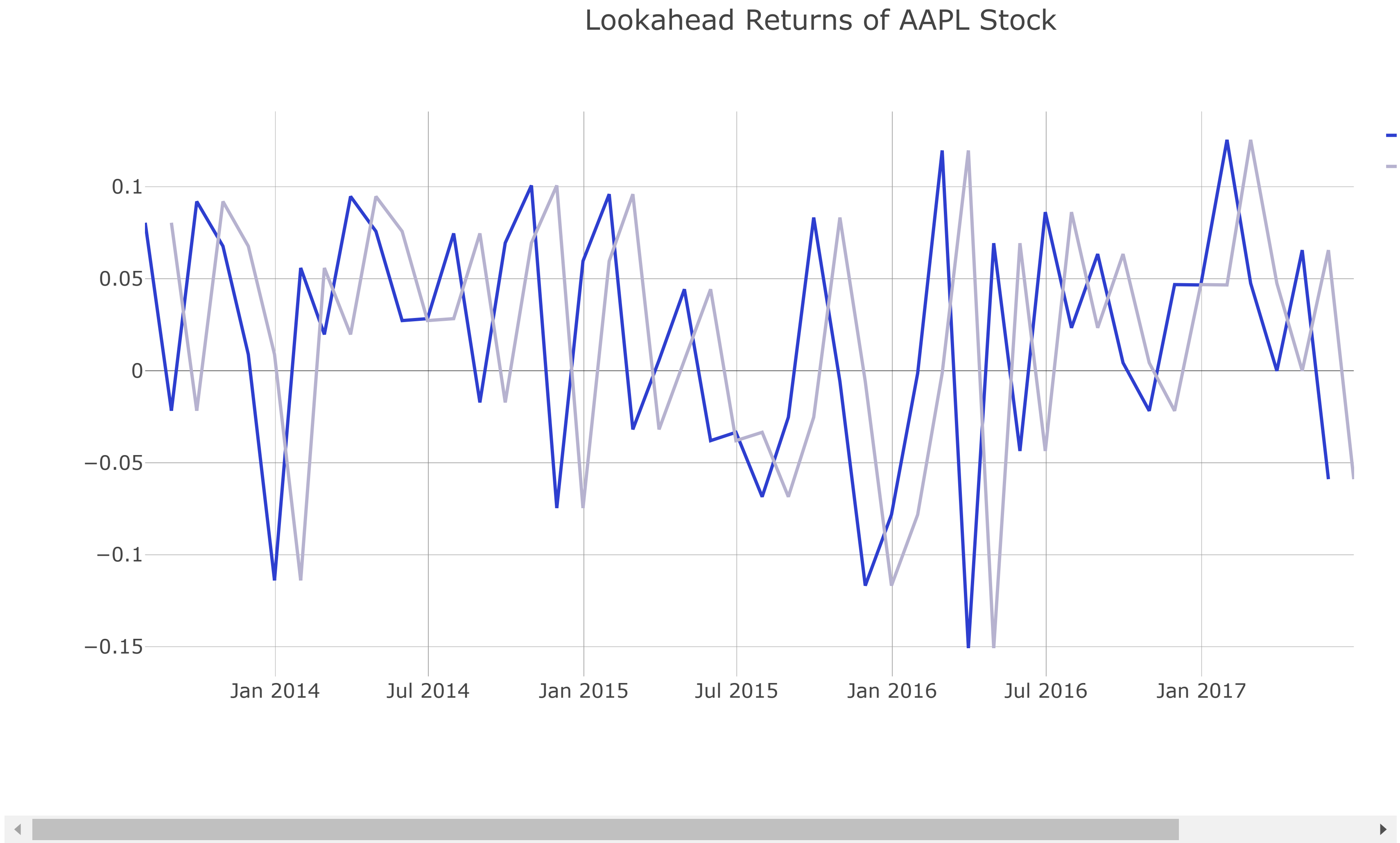
```
Tests Passed
```

View Data

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

In [60]:
```
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))
```


Previous Returns of AAPL Stock
0.1
0.05
0
−0.05
−0.1
−0.15
Jan 2014
Jul 2014
Jan 2015
Jul 2015
Jan 2016
Jul 2016
Jan 2017


Lookahead Returns of AAPL Stock
0.1
0.05
0
−0.05
−0.1
−0.15
Jan 2014
Jul 2014
Jan 2015
Jul 2015
Jan 2016
Jul 2016
Jan 2017

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`](https://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.DataFrame.iterrows.html) (https://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.DataFrame.iterrows.html) with [`Series.nlargest`](https://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.Series.nlargest.html) (https://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.Series.nlargest.html) in order to implement the function. This is one of those cases where creating a vecorization solution is too difficult.

In [158]:
```
#step1 get some sample of data
a=prev_returns.tail(10).T.tail(10).T
a
```

Out[158]:

ticker date	XL	XLNX	XOM	XRAY	XRX	XYL	YUM	ZBH	ZION	ZTS
2016-09-30	-0.01104021	0.06579968	-0.01191629	-0.04112133	-0.04467244	0.06487578	0.01432323	-0.01173551	0.09555297	0.01240540
2016-10-31	-0.01178072	0.00239521	0.00160532	-0.03228879	0.03568895	0.03078357	0.00110181	0.00504351	0.01395900	0.01765151
2016-11-30	0.03132116	-0.06598725	-0.04643273	-0.03179755	-0.03618485	-0.08180785	-0.04530849	-0.20992565	0.03764481	-0.08441037
2016-12-31	0.04038386	0.06594093	0.05541489	0.01054005	-0.04394012	0.06826631	0.02387708	-0.03416331	0.21341820	0.05457719
2017-01-31	0.03615589	0.11187869	0.03334393	-0.00643281	-0.06031217	-0.04075737	-0.00094697	0.01539556	0.07851429	0.06064797
2017-02-28	0.00828549	-0.03660694	-0.07318798	-0.01800276	0.14387403	-0.00424973	0.03879000	0.13680848	-0.01994669	0.02796748
2017-03-31	0.07484917	0.01624776	-0.02195182	0.11358061	0.07101104	-0.02065858	-0.00320978	-0.01061987	0.06402762	-0.03011775
2017-04-30	-0.01017620	-0.01593728	0.00844920	-0.01575947	-0.00497178	0.04271546	-0.02197891	0.04404029	-0.06676818	0.00112486
2017-05-31	0.04871848	0.08633458	-0.00439937	0.01273092	-0.02064767	0.02341935	0.03320595	-0.02035146	-0.04804045	0.05205758
2017-06-30	0.04302745	0.06090411	-0.00482798	0.00441780	-0.01683069	0.01750300	0.09965606	-0.00368417	0.00296435	0.10432630

In [159]:

```
#step 2, finding maximum of numbers in different columns
top_a = a.apply(lambda x: x.nlargest(3), axis=1)
top_a
```

Out[159]:

date	XL	XLNX	XOM	XRAY	XRX	XYL	YUM	ZBH	ZION	ZTS
2016-09-30	nan	0.06579968	nan	nan	nan	0.06487578	nan	nan	0.09555297	nan
2016-10-31	nan	nan	nan	nan	0.03568895	0.03078357	nan	nan	nan	0.01765151
2016-11-30	0.03132116	nan	nan	-0.03179755	nan	nan	nan	nan	0.03764481	nan
2016-12-31	nan	0.06594093	nan	nan	nan	0.06826631	nan	nan	0.21341820	nan
2017-01-31	nan	0.11187869	nan	nan	nan	nan	nan	nan	0.07851429	0.06064797
2017-02-28	nan	nan	nan	nan	0.14387403	nan	0.03879000	0.13680848	nan	nan
2017-03-31	0.07484917	nan	nan	0.11358061	0.07101104	nan	nan	nan	nan	nan
2017-04-30	nan	nan	0.00844920	nan	nan	0.04271546	nan	0.04404029	nan	nan
2017-05-31	0.04871848	0.08633458	nan	nan	nan	nan	nan	nan	nan	0.05205758
2017-06-30	nan	0.06090411	nan	nan	nan	nan	0.09965606	nan	nan	0.10432630

In [160]:
```
#step 3 mapping nulls to 0 and the rest to 1 using applymap
top_a = top_a.applymap(lambda x: 0 if pd.isna(x) else 1)
top_a = top_a.astype(np.int64)
top_a
```

Out[160]:

	XL	XLNX	XOM	XRAY	XRX	XYL	YUM	ZBH	ZION	ZTS
date										
2016-09-30	0	1	0	0	0	1	0	0	1	0
2016-10-31	0	0	0	0	1	1	0	0	0	1
2016-11-30	1	0	0	1	0	0	0	0	1	0
2016-12-31	0	1	0	0	0	1	0	0	1	0
2017-01-31	0	1	0	0	0	0	0	0	1	1
2017-02-28	0	0	0	0	1	0	1	1	0	0
2017-03-31	1	0	0	1	1	0	0	0	0	0
2017-04-30	0	0	1	0	0	1	0	1	0	0
2017-05-31	1	1	0	0	0	0	0	0	0	1
2017-06-30	0	1	0	0	0	0	1	0	0	1

In [162]:
```
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
    top_a = prev_returns.apply(lambda x: x.nlargest(top_n), axis=1)
    top_a = top_a.applymap(lambda x: 0 if pd.isna(x) else 1)
    return top_a.astype(np.int64)

project_tests.test_get_top_n(get_top_n)
```

```
Tests Passed
```

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 [163]:
```
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:
INCY, AMD, AVGO, NFX, NFLX, SWKS, ILMN, NVDA, UAL, MU
10 Most Shorted Stocks:
RRC, CHK, FCX, MRO, FTI, DVN, GPS, WYNN, SPLS, MAT
```

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 [170]:
```
def portfolio_returns(df_long, df_short, lookahead_returns, n_stocks):
    """
    Compute expected returns for the portfolio, assuming equal investment in each long/short stock.

    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

    return (df_long - df_short)*lookahead_returns / n_stocks

project_tests.test_portfolio_returns(portfolio_returns)
```

```
Tests Passed
```

View Data

Time to see how the portfolio did.

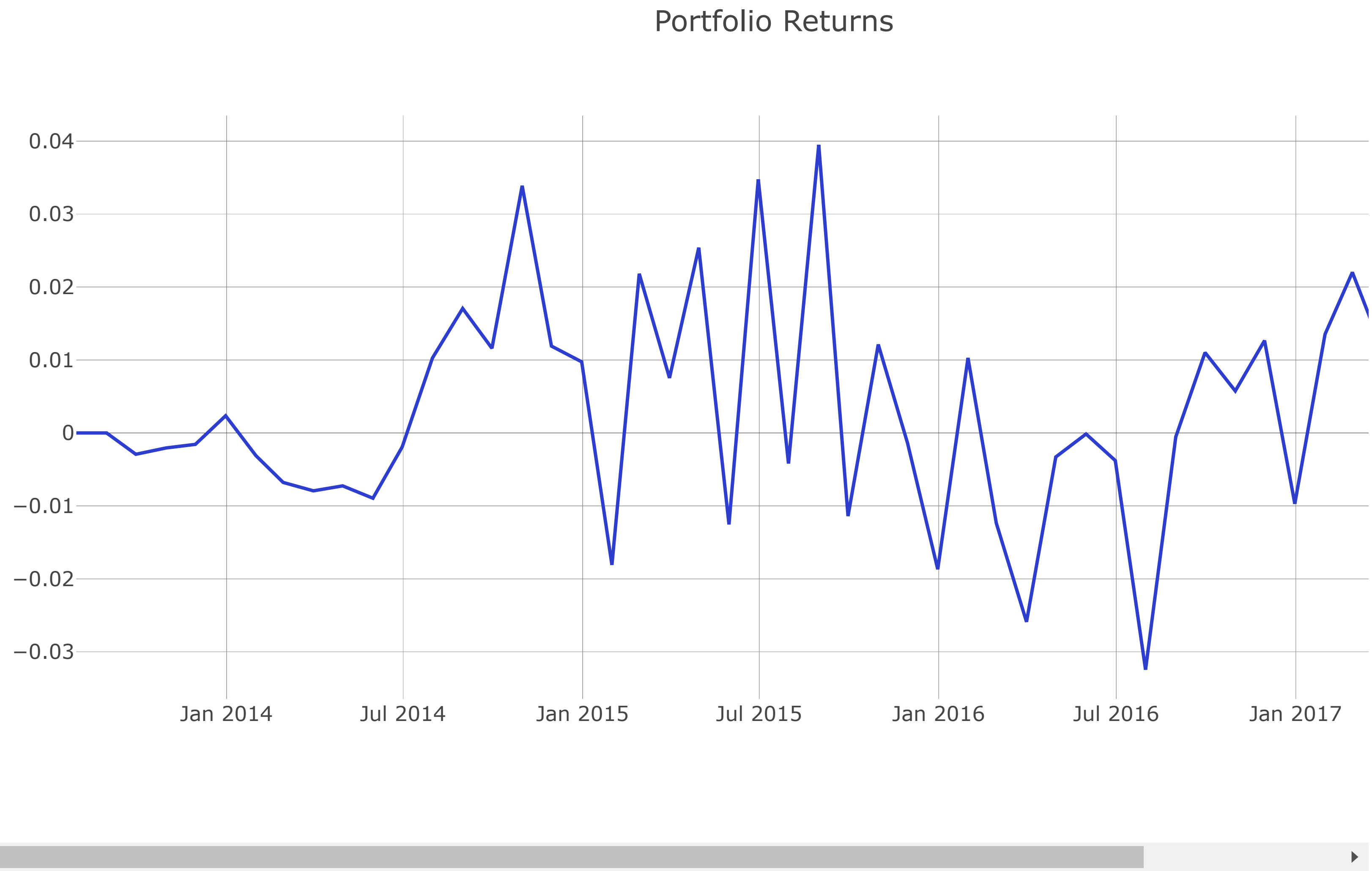
In [167]:
```
df_long.shape,df_short.shape
```

Out[167]:
```
((48, 457), (48, 470))
```

In [168]:
```
top_bottom_n
```

Out[168]:
```
50
```

In [171]:
```
expected_portfolio_returns = portfolio_returns(df_long, df_short, lookahead_returns, 2*top_bottom_n)
project_helper.plot_returns(expected_portfolio_returns.T.sum(), 'Portfolio Returns')
```



Statistical Tests

Annualized Rate of Return

In [173]:
```
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:                        0.003076
Standard Error:              0.002180
Annualized Rate of Return:   3.76%
```

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.

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` (https://docs.scipy.org/doc/scipy-1.0.0/reference/generated/scipy.stats.ttest_1samp.html)](https://docs.scipy.org/doc/scipy-1.0.0/reference/generated/scipy.stats.ttest_1samp.html) performs a two-sided test, so divide the p-value by 2 to get 1-sided p-value

In [180]:
```
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 zero.

    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 #because the function is two sided is devided by two

project_tests.test_analyze_alpha(analyze_alpha)
```

```
Tests Passed
```

View Data

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

In [181]:
```
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:        1.411
 p-value:        0.082517
```

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

#TODO: Put Answer In this Cell

our p-value is .082517, and since it is greater thatn the alpha=.05, this tells us that Annualized Rate of Return: 3.76% that we have seen is very likely to happen. so this rate of return is just a high probability chance and we cant rely on it.

In fact we want lower p-values to go with this strategy, because then it tells us it would be unlikely to observe a high rate of return, and since it has happened then it is probably not by chance.

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.