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 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.

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))
Collecting cvxpy==1.0.3 (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/a1/59/2613468ffbbe3
a818934d06b81b9f4877fe054afbf4f99d2f43f398a0b34/cvxpy-1.0.3.tar.gz (880k
B)
    100% |████████████████████████████████| 880kB 462kB/s ta 0:00:011
Requirement already satisfied: cycler==0.10.0 in /opt/conda/lib/python3.
6/site-packages/cycler-0.10.0-py3.6.egg (from -r requirements.txt (line
3))
Collecting numpy==1.13.3 (from -r requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/57/a7/e3e6bd9d59512
5e1abbe162e323fd2d06f6f6683185294b79cd2cdb190d5/numpy-1.13.3-cp36-cp36m-m
anylinux1_x86_64.whl (17.0MB)
    100% |████████████████████████████████| 17.0MB 27kB/s  eta 0:00:01
```

```
4% |▌                               | 686kB 30.6MB/s eta 0:00:01
Collecting pandas==0.21.1 (from -r requirements.txt (line 5))
  Downloading https://files.pythonhosted.org/packages/3a/e1/6c514df670b88
7c77838ab856f57783c07e8760f2e3d5939203a39735e0e/pandas-0.21.1-cp36-cp36m-
manylinux1_x86_64.whl (26.2MB)
    100% |████████████████████████████████| 26.2MB 17kB/s  eta 0:00:01
52% |████████████████▊               | 13.7MB 31.3MB/s eta 0:00:01    63%
|████████████████████▏           | 16.6MB 29.9MB/s eta 0:00:01    73% |██
██████████████████████▍         | 19.4MB 26.4MB/s eta 0:00:01    85% |█████
██████████████████████████      | 22.4MB 29.9MB/s eta 0:00:01
Collecting plotly==2.2.3 (from -r requirements.txt (line 6))
  Downloading https://files.pythonhosted.org/packages/99/a6/8214b6564bf4a
ce9bec8a26e7f89832792be582c042c47c912d3201328a0/plotly-2.2.3.tar.gz (1.1M
B)
    100% |████████████████████████████████| 1.1MB 436kB/s eta 0:00:01
72% |███████████████████████         | 788kB 30.0MB/s eta 0:00:01
Requirement already satisfied: pyparsing==2.2.0 in /opt/conda/lib/python
3.6/site-packages (from -r requirements.txt (line 7))
Requirement already satisfied: python-dateutil==2.6.1 in /opt/conda/lib/p
ython3.6/site-packages (from -r requirements.txt (line 8))
Requirement already satisfied: pytz==2017.3 in /opt/conda/lib/python3.6/s
ite-packages (from -r requirements.txt (line 9))
Requirement already satisfied: requests==2.18.4 in /opt/conda/lib/python
3.6/site-packages (from -r requirements.txt (line 10))
Collecting scipy==1.0.0 (from -r requirements.txt (line 11))
  Downloading https://files.pythonhosted.org/packages/d8/5e/caa01ba7be116
00b6a9d39265440d7b3be3d69206da887c42bef049521f2/scipy-1.0.0-cp36-cp36m-ma
nylinux1_x86_64.whl (50.0MB)
    100% |████████████████████████████████| 50.0MB 9.1kB/s eta 0:00:01
6% |▋                               | 3.2MB 26.4MB/s eta 0:00:02    11% |
███▏                            | 5.7MB 26.3MB/s eta 0:00:02    14% |████
▊                           | 7.0MB 26.4MB/s eta 0:00:02    18% |██████
| 9.3MB 23.0MB/s eta 0:00:02    21% |███████                         | 1
0.7MB 27.6MB/s eta 0:00:02    23% |████████                        | 11.9
MB 26.2MB/s eta 0:00:02    28% |█████████                       | 14.2MB
26.4MB/s eta 0:00:02    30% |██████████                      | 15.5MB 25.
1MB/s eta 0:00:02    38% |████████████                    | 19.1MB 26.1M
B/s eta 0:00:02    47% |███████████████                 | 23.9MB 24.5MB/s
eta 0:00:02    52% |████████████████                | 26.3MB 28.6MB/s eta
0:00:01    55% |█████████████████               | 27.6MB 27.1MB/s eta 0:0
0:01    57% |██████████████████              | 28.9MB 25.7MB/s eta 0:00:0
1    60% |███████████████████             | 30.2MB 28.5MB/s eta 0:00:01
65% |████████████████████            | 32.7MB 27.8MB/s eta 0:00:01    70%
|██████████████████████          | 35.1MB 24.9MB/s eta 0:00:01    72% |██
██████████████████████          | 36.3MB 29.2MB/s eta 0:00:01    75% |█████
███████████████████         | 37.7MB 23.2MB/s eta 0:00:01    80% |███████
████████████████        | 40.1MB 30.0MB/s eta 0:00:01    84% |████████████
██████████████      | 42.5MB 27.3MB/s eta 0:00:01    87% |████████████████
████████████▏     | 43.6MB 21.9MB/s eta 0:00:01    89% |████████████████
████████████    | 44.8MB 28.6MB/s eta 0:00:01    92% |████████████████████
██████████   | 46.2MB 31.3MB/s eta 0:00:01    99% |████████████████████████
████████| 49.5MB 19.7MB/s eta 0:00:01
Requirement already satisfied: scikit-learn==0.19.1 in /opt/conda/lib/pyt
hon3.6/site-packages (from -r requirements.txt (line 12))
Requirement already satisfied: six==1.11.0 in /opt/conda/lib/python3.6/si
te-packages (from -r requirements.txt (line 13))
Collecting tqdm==4.19.5 (from -r requirements.txt (line 14))
```

```
  Downloading https://files.pythonhosted.org/packages/71/3c/341b4fa23cb3a
bc335207dba057c790f3bb329f6757e1fcd5d347bcf8308/tqdm-4.19.5-py2.py3-none-
any.whl (51kB)
    100% |████████████████████████████████| 61kB 3.8MB/s eta 0:00:01
Collecting osqp (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/c0/01/8becb29b0d38e
0c40eab9e3d54aa8138fa62a010d519caf65e9210021bd3/osqp-0.5.0-cp36-cp36m-man
ylinux1_x86_64.whl (147kB)
    100% |████████████████████████████████| 153kB 2.5MB/s eta 0:00:01
Collecting ecos>=2 (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/55/ed/d131ff51f3a8f
73420eb1191345eb49f269f23cadef515172e356018cde3/ecos-2.0.7.post1-cp36-cp3
6m-manylinux1_x86_64.whl (147kB)
    100% |████████████████████████████████| 153kB 2.8MB/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/6e01c4f4a69fc
c6c3db130ba55572089e78e77ea8c0921a679f9da1ec04c/scs-2.0.2.tar.gz (133kB)
    100% |████████████████████████████████| 143kB 2.7MB/s eta 0:00:01
Collecting multiprocess (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/31/60/6d74caa02b54c
a43092e745640c7d98f367f07160441682a01602ce00bc5/multiprocess-0.70.7.tar.g
z (1.4MB)
    100% |████████████████████████████████| 1.4MB 337kB/s eta 0:00:01
36% |████████████                            | 512kB 21.9MB/s eta 0:00:01
Requirement already satisfied: fastcache in /opt/conda/lib/python3.6/site
-packages (from cvxpy==1.0.3->-r requirements.txt (line 2))
Requirement already satisfied: toolz in /opt/conda/lib/python3.6/site-pac
kages (from cvxpy==1.0.3->-r requirements.txt (line 2))
Requirement already satisfied: decorator>=4.0.6 in /opt/conda/lib/python
3.6/site-packages (from plotly==2.2.3->-r requirements.txt (line 6))
Requirement already satisfied: nbformat>=4.2 in /opt/conda/lib/python3.6/
site-packages (from plotly==2.2.3->-r requirements.txt (line 6))
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/py
thon3.6/site-packages (from requests==2.18.4->-r requirements.txt (line 1
0))
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))
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /opt/conda/lib/py
thon3.6/site-packages (from requests==2.18.4->-r requirements.txt (line 1
0))
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/pytho
n3.6/site-packages (from requests==2.18.4->-r requirements.txt (line 10))
Requirement already satisfied: future in /opt/conda/lib/python3.6/site-pa
ckages (from osqp->cvxpy==1.0.3->-r requirements.txt (line 2))
Collecting dill>=0.2.9 (from multiprocess->cvxpy==1.0.3->-r requirements.
txt (line 2))
  Downloading https://files.pythonhosted.org/packages/fe/42/bfe2e0857bc28
4cbe6a011d93f2a9ad58a22cb894461b199ae72cfef0f29/dill-0.2.9.tar.gz (150kB)
    100% |████████████████████████████████| 153kB 2.5MB/s eta 0:00:01
Requirement already satisfied: jupyter-core in /opt/conda/lib/python3.6/s
ite-packages (from nbformat>=4.2->plotly==2.2.3->-r requirements.txt (lin
e 6))
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))
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /opt/conda/lib/
python3.6/site-packages (from nbformat>=4.2->plotly==2.2.3->-r requiremen
```

```
ts.txt (line 6))
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python
3.6/site-packages (from nbformat>=4.2->plotly==2.2.3->-r requirements.txt
(line 6))
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/0c2596528665e21d6
98d6f84a3406c52044c7b4ca6ac737cf3
  Running setup.py bdist_wheel for plotly ... done
  Stored in directory: /root/.cache/pip/wheels/98/54/81/dd92d5b0858fac680
cd7bdb8800eb26c001dd9f5dc8b1bc0ba
  Running setup.py bdist_wheel for scs ... done
  Stored in directory: /root/.cache/pip/wheels/ff/f0/aa/530ccd478d7d9900b
4e9ef5bc5a39e895ce110bed3d3ac653e
  Running setup.py bdist_wheel for multiprocess ... done
  Stored in directory: /root/.cache/pip/wheels/3a/ed/51/77c833462c3e757ce
50c4b2b68bdf53f5d1745542fe567d740
  Running setup.py bdist_wheel for dill ... done
  Stored in directory: /root/.cache/pip/wheels/5b/d7/0f/e58eae695403de585
269f4e4a94e0cd6ca60ec0c202936fa4a
Successfully built cvxpy plotly scs multiprocess dill
Installing collected packages: numpy, scipy, osqp, ecos, scs, dill, multi
process, 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 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-0.2.9 ecos-2.0.7.post1 multiproce
ss-0.70.7 numpy-1.13.3 osqp-0.5.0 pandas-0.21.1 plotly-2.2.3 scipy-1.0.0
scs-2.0.2 tqdm-4.19.5
You are using pip version 9.0.1, however version 19.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' comman
d.
```

Load Packages

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=[

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

print('Loaded Data')
```

```
Loaded Data
```

View Data

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

In [3]:
```
# Do not use as the project_helper file contains a but with plotly
# See Udacity Forums, Trading with Momentum, Thread by JACOB V. and
# answer by mentor IoannisK at 26.12.2018 9:00 AM
#project_helper.print_dataframe(close)

# Use normal close.head() to show data
close.head()
```

Out[3]:

ticker	A	AAL	AAP	AAPL	ABBV	ABC
date						
2013-07-01	29.99418563	16.17609308	81.13821681	53.10917319	34.92447839	50.86319750
2013-07-02	29.65013670	15.81983388	80.72207258	54.31224742	35.42807578	50.69676639
2013-07-03	29.70518453	16.12794994	81.23729877	54.61204262	35.44486235	50.93716689
2013-07-05	30.43456826	16.21460758	81.82188233	54.17338125	35.85613355	51.37173702
2013-07-08	30.52402098	16.31089385	82.95141667	53.86579916	36.66188936	52.03746147

5 rows × 495 columns

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 [4]:
```
apple_ticker = 'AAPL'
project_helper.plot_stock(close[apple_ticker], '{} Stock'.format(apple_ti
```

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 [5]:
```
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/

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

project_tests.test_resample_prices(resample_prices)
```

```
Tests Passed
```

View Data

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

In [6]:
```
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 to help you calculate the log returns.

In [7]:
```
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 [8]:
```
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))
monthly_close_returns.head()
```

Out[8]:

ticker date	A	AAL	AAP	AAPL	ABBV	ABC	
2013-07-31	nan	nan	nan	nan	nan	nan	
2013-08-31	0.04181412	-0.18015337	-0.02977582	0.08044762	-0.06518370	-0.01609335	-0
2013-09-30	0.09657861	0.15979244	0.03282284	-0.02171531	0.04855545	0.07086509	-0
2013-10-31	-0.00960698	0.14734639	0.18195865	0.09201927	0.08860637	0.06693948	0
2013-11-30	0.05388057	0.06647111	0.01828314	0.06772063	0.00000000	0.07997603	0

5 rows × 495 columns

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 [9]:
```
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 [10]:
```
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))
```

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 [11]:
```
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_stocks = pd.DataFrame(columns = prev_returns.columns.values, inde

    for index, row in prev_returns.iterrows():
        nlargest_array = np.array(row.nlargest(top_n))
        row_list = list()

        for item in row:
            row_list.append(item in nlargest_array)

        top_stocks.loc[index] = pd.Series(np.array(row_list).astype(np.in

    return top_stocks

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 [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')
```

```
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
```

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 [18]:
```
def portfolio_returns(df_long, df_short, lookahead_returns, n_stocks):
    """
    Compute expected returns for the portfolio, assuming equal investment

    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 (lookahead_returns * df_long / n_stocks) - (lookahead_returns

project_tests.test_portfolio_returns(portfolio_returns)
```

```
Tests Passed
```

View Data

Time to see how the portfolio did.

In [19]:
```
expected_portfolio_returns = portfolio_returns(df_long, df_short, lookahe
project_helper.plot_returns(expected_portfolio_returns.T.sum(), 'Portfoli
```

Statistical Tests

Annualized Rate of Return

In [20]:
```
expected_portfolio_returns_by_date = expected_portfolio_returns.T.sum().d
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_ra
```

```
Mean:                         0.003253
Standard Error:               0.002203
Annualized Rate of Return:  3.98%
```

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` performs a two-sided test, so divide the p-value by 2 to get 1-sided p-value

In [24]:
```
from scipy import stats

def analyze_alpha(expected_portfolio_returns_by_date):
    """
    Perform a t-test with the null hypothesis being that the expected mea

    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
```

View Data

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

In [25]:
```
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:        0.073359
```

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

#TODO: Put Answer In this Cell

Under the assumption that the null hypothesis is true, our portfolio returns a p-value of 0.073359 which indicates the probability to have a annual rate of return of 3.98% as we calculated above.

As 0.073359 is greater than the alpha of 0.05 that we have selected as the level of our significance test, we cannot conclude that the null hypothesis is to be rejected.

A p-value of 0.073359 is equivalent to a probability of ca. 7.3% that we find a rate of return of 3.98% under the condition that the null hypothesis is true, that is, that the real mean of our portfolio is 0%.

This is a relatively high probability, and so gives us no real device to conclude that the null hypothesis is not true. If, for example, we had a p-value of, say, 0.02 or equivalently a probability of 2%, this would let the annual return of 3.98% quite unprobable under the condition that the null hypothesis is true; we would then easily conclude that the null hypothesis is false.

As we have in advance decicded on the treshold of alpha = 0.05, and the p-value of 0.073359 that we found is greater

Hence the null hypothesis can count as being still valid, which means that the annual rate of return is 0%.

Hence we should not invest into the portfolio as calculated above.

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.