Project 2: Breakout Strategy

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/) and Numpy (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 6.9MB/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 2.3MB/s eta 0:00:01  3% |                                |
532kB 11.2MB/s eta 0:00:02    23% |███████                         | 4.0MB 23.0MB/s eta 0:00:01    31% |█████
████                          | 5.3MB 26.8MB/s eta 0:00:01    47% |██████████████                  | 8.0MB 27.2M
B/s eta 0:00:01    55% |█████████████████               | 9.3MB 27.0MB/s eta 0:00:01    62% |████████████
████              | 10.7MB 26.7MB/s eta 0:00:01    86% |███████████████████████████     | 14.6MB 28.2MB/s eta
0:00:01    93% |█████████████████████████████   | 15.9MB 26.7MB/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 1.6MB/s eta 0:00:01  3% |▉
| 931kB 27.0MB/s eta 0:00:01    14% |████▉                           | 3.7MB 26.7MB/s eta 0:00:01    19% |██
████                          | 5.1MB 28.8MB/s eta 0:00:01    30% |█████████▉                              | 7.9MB
27.7MB/s eta 0:00:01    61% |███████████████████▉            | 16.1MB 27.6MB/s eta 0:00:01    66% |████████
████████████              | 17.4MB 26.3MB/s eta 0:00:01    71% |██████████████████████▉         | 18.7MB 28.8M
B/s eta 0:00:01    81% |█████████████████████████       | 21.3MB 26.9MB/s eta 0:00:01
Collecting plotly==2.2.3 (from -r requirements.txt (line 6))
  Downloading https://files.pythonhosted.org/packages/99/a6/8214b6564bf4ace9bec8a26e7f89832792be582c042c47c91
2d3201328a0/plotly-2.2.3.tar.gz (1.1MB)
    100% |████████████████████████████████| 1.1MB 10.4MB/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 711kB/s eta 0:00:01  8% |██▉
| 4.1MB 25.5MB/s eta 0:00:02    15% |████▉                           | 7.7MB 25.1MB/s eta 0:00:02    17% |███
```

```
██▏                                | 8.7MB 20.9MB/s eta 0:00:02    19% |██████▏                                 | 9.9MB 2
3.0MB/s eta 0:00:02    24% |███████▍                              | 12.2MB 22.7MB/s eta 0:00:02    26% |████████▏
| 13.5MB 23.8MB/s eta 0:00:02    28% |████████▋                             | 14.5MB 25.3MB/s eta 0:00:02    35% |
███████████▏                           | 17.9MB 21.8MB/s eta 0:00:02    42% |█████████████▏                          | 21.
4MB 24.9MB/s eta 0:00:02    47% |███████████████                       | 23.6MB 25.2MB/s eta 0:00:02    50% |██████
██████████                     | 25.4MB 27.0MB/s eta 0:00:01    53% |████████████████                  | 26.7MB 23.
6MB/s eta 0:00:01    57% |██████████████████▏                   | 28.8MB 25.0MB/s eta 0:00:01    59% |████████████
███████                | 29.9MB 23.1MB/s eta 0:00:01    61% |███████████████████▏                | 31.0MB 24.3MB/s
eta 0:00:01    64% |████████████████████▏                 | 32.2MB 25.3MB/s eta 0:00:01    71% |██████████████████
████▏           | 35.5MB 23.1MB/s eta 0:00:01    73% |██████████████████████           | 36.8MB 24.5MB/s eta 0:
00:01    75% |███████████████████████▏          | 38.0MB 21.5MB/s eta 0:00:01    82% |████████████████████████
██▏       | 41.1MB 23.9MB/s eta 0:00:01    84% |██████████████████████████      | 42.4MB 24.8MB/s eta 0:00:01
89% |████████████████████████████    | 44.6MB 27.8MB/s eta 0:00:01    93% |████████████████████████████████  |
46.8MB 25.0MB/s eta 0:00:01    96% |██████████████████████████████▏ | 48.1MB 25.0MB/s eta 0:00:01    98% |██
████████████████████████████▏ | 49.1MB 21.5MB/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 10.4MB/s ta 0:00:01
Collecting zipline==1.2.0 (from -r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/15/d3/689f2a940478b82ac57c751a40460598221fd82b0449a7a8f
7eef47a3bcc/zipline-1.2.0.tar.gz (659kB)
    100% |████████████████████████████████| 665kB 15.5MB/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 16.7MB/s ta 0:00:01
Collecting ecos>=2 (from cvxpy==1.0.3->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/55/ed/d131ff51f3a8f73420eb1191345eb49f269f23cadef515172
e356018cde3/ecos-2.0.7.post1-cp36-cp36m-manylinux1_x86_64.whl (147kB)
    100% |████████████████████████████████| 153kB 16.7MB/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/1a/72/33be87cce255d4e9dbbfef547e9fd6ec7ee94d0d0910bb2b1
3badea3fbbe/scs-2.1.2.tar.gz (3.5MB)
    100% |████████████████████████████████| 3.6MB 8.5MB/s eta 0:00:01    26% |████████▏
| 952kB 20.4MB/s eta 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 8.3MB/s eta 0:00:01   7% |██▉
| 112kB 13.7MB/s eta 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: pip>=7.1.0 in /opt/conda/lib/python3.6/site-packages (from zipline==1.2.0->-r
requirements.txt (line 15)) (18.1)
Requirement already satisfied: setuptools>18.0 in /opt/conda/lib/python3.6/site-packages (from zipline==1.2.0
->-r requirements.txt (line 15)) (38.4.0)
Collecting Logbook>=0.12.5 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/2f/d9/16ac346f7c0102835814cc9e5b684aaadea101560bb932a24
03bd26b2320/Logbook-1.5.3.tar.gz (85kB)
    100% |████████████████████████████████| 92kB 9.7MB/s eta 0:00:01
Collecting requests-file>=1.4.1 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/23/9c/6e63c23c39e53d3df41c77a3d05a49a42c4e1383a6d2a5e32
33161b89dbf/requests_file-1.4.3-py2.py3-none-any.whl
Collecting pandas-datareader<0.6,>=0.2.1 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/40/c5/cc720f531bbde0efeab940de400d0fcc95e87770a3abcd7f9
0d6d52a3302/pandas_datareader-0.5.0-py2.py3-none-any.whl (74kB)
    100% |████████████████████████████████| 81kB 5.9MB/s ta 0:00:01
Requirement already satisfied: patsy>=0.4.0 in /opt/conda/lib/python3.6/site-packages (from zipline==1.2.0->-
r requirements.txt (line 15)) (0.4.1)
Requirement already satisfied: statsmodels>=0.6.1 in /opt/conda/lib/python3.6/site-packages (from zipline==1.
2.0->-r requirements.txt (line 15)) (0.8.0)
Requirement already satisfied: Cython>=0.25.2 in /opt/conda/lib/python3.6/site-packages (from zipline==1.2.0-
>-r requirements.txt (line 15)) (0.29.7)
Collecting cyordereddict>=0.2.2 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/d1/1a/364cbfd927be1b743c7f0a985a7f1f7e8a51469619f9fefe4
ee9240ba210/cyordereddict-1.0.0.tar.gz (138kB)
    100% |████████████████████████████████| 143kB 15.3MB/s ta 0:00:01
```

```
Collecting bottleneck>=1.0.0 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/5b/08/278c6ee569458e168096f6b51019cc1c81c288da3d1026a22
ee2ccead102/Bottleneck-1.3.2.tar.gz (88kB)
    100% |████████████████████████████████| 92kB 12.6MB/s ta 0:00:01
  Installing build dependencies ... done
Collecting contextlib2>=0.4.0 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/85/60/370352f7ef6aa96c52fb001831622f50f923c1d575427d021
b8ab3311236/contextlib2-0.6.0.post1-py2.py3-none-any.whl
Requirement already satisfied: networkx<2.0,>=1.9.1 in /opt/conda/lib/python3.6/site-packages (from zipline==
1.2.0->-r requirements.txt (line 15)) (1.11)
Requirement already satisfied: numexpr>=2.6.1 in /opt/conda/lib/python3.6/site-packages (from zipline==1.2.0-
>-r requirements.txt (line 15)) (2.6.4)
Collecting bcolz<1,>=0.12.1 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/6c/8b/1ffa01f872cac36173c5eb95b58c01040d8d25f1b242c4857
7f4104cd3ab/bcolz-0.12.1.tar.gz (622kB)
    100% |████████████████████████████████| 624kB 14.8MB/s ta 0:00:01
Requirement already satisfied: click>=4.0.0 in /opt/conda/lib/python3.6/site-packages (from zipline==1.2.0->-
r requirements.txt (line 15)) (6.7)
Collecting multipledispatch>=0.4.8 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/89/79/429ecef45fd5e4504f7474d4c3c3c4668c267be3370e4c2fd
33e61506833/multipledispatch-0.6.0-py3-none-any.whl
Requirement already satisfied: MarkupSafe>=0.23 in /opt/conda/lib/python3.6/site-packages (from zipline==1.2.
0->-r requirements.txt (line 15)) (1.0)
Requirement already satisfied: Mako>=1.0.1 in /opt/conda/lib/python3.6/site-packages/Mako-1.0.7-py3.6.egg (fr
om zipline==1.2.0->-r requirements.txt (line 15)) (1.0.7)
Requirement already satisfied: sqlalchemy>=1.0.8 in /opt/conda/lib/python3.6/site-packages (from zipline==1.
2.0->-r requirements.txt (line 15)) (1.1.13)
Collecting alembic>=0.7.7 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/60/1e/cabc75a189de0fbb2841d0975243e59bde8b7822bacbb9500
8ac6fe9ad47/alembic-1.4.2.tar.gz (1.1MB)
    100% |████████████████████████████████| 1.1MB 14.3MB/s ta 0:00:01    18% |██████
| 204kB 18.2MB/s eta 0:00:01
  Installing build dependencies ... done
Collecting sortedcontainers>=1.4.4 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/13/f3/cf85f7c3a2dbd1a515d51e1f1676d971abe41bba6f4ab5443
240d9a78e5b/sortedcontainers-2.1.0-py2.py3-none-any.whl
Collecting intervaltree>=2.1.0 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/e8/f9/76237755b2020cd74549e98667210b2dd54d3fb17c6f4a626
31e61d31225/intervaltree-3.0.2.tar.gz
Collecting lru-dict>=1.1.4 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/00/a5/32ed6e10246cd341ca8cc205acea5d208e4053f48a4dced2b
1b31d45ba3f/lru-dict-1.1.6.tar.gz
Collecting empyrical>=0.4.2 (from zipline==1.2.0->-r requirements.txt (line 15))
```

```
  Downloading https://files.pythonhosted.org/packages/84/9e/9506e8b25464ff57ef93b5ba9092b464b44dc76b717695b12
6b3c93214a2/empyrical-0.5.3.tar.gz (50kB)
    100% |████████████████████████████████| 51kB 4.8MB/s ta 0:00:01    81% |██████████████████████████      |
40kB 9.1MB/s eta 0:00:01
Collecting tables>=3.3.0 (from zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/ed/c3/8fd9e3bb21872f9d69eb93b3014c86479864cca94e625fd03
713ccacec80/tables-3.6.1-cp36-cp36m-manylinux1_x86_64.whl (4.3MB)
    100% |████████████████████████████████| 4.3MB 7.9MB/s eta 0:00:01    25% |████████
| 1.1MB 22.6MB/s eta 0:00:01    72% |███████████████████████         | 3.1MB 19.8MB/s eta 0:00:01
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 15.6MB/s ta 0:00:01
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: 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)
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: 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)
Collecting requests-ftp (from pandas-datareader<0.6,>=0.2.1->zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/3d/ca/14b2ad1e93b5195eeaf56b86b7ecfd5ea2d5754a68d17aeb1
e5b9f95b3cf/requests-ftp-0.3.1.tar.gz
Collecting python-editor>=0.3 (from alembic>=0.7.7->zipline==1.2.0->-r requirements.txt (line 15))
  Downloading https://files.pythonhosted.org/packages/c6/d3/201fc3abe391bbae6606e6f1d598c15d367033332bd54352b
12f35513717/python_editor-1.0.4-py3-none-any.whl
Building wheels for collected packages: cvxpy, plotly, zipline, scs, multiprocess, Logbook, cyordereddict, bo
ttleneck, bcolz, alembic, intervaltree, lru-dict, empyrical, dill, requests-ftp
  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 zipline ... done
  Stored in directory: /root/.cache/pip/wheels/5d/20/7d/b48368c8634b1cb6cc7232833b2780a265d4217c0ad2e3d24c
  Running setup.py bdist_wheel for scs ... done
  Stored in directory: /root/.cache/pip/wheels/df/d0/79/37ea880586da03c620ca9ecd5e42adbd86bc6ea84363965c5f
  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 Logbook ... done
  Stored in directory: /root/.cache/pip/wheels/d2/70/07/68b99a8e05dcd1ab194a8e0ccb9e4d0ac5dd6d8d139c7149b4
```

```
  Running setup.py bdist_wheel for cyordereddict ... done
  Stored in directory: /root/.cache/pip/wheels/0b/9d/8b/5bf3e22c1edd59b50f11bb19dec9dfcfe5a479fc7ace02b61f
  Running setup.py bdist_wheel for bottleneck ... done
  Stored in directory: /root/.cache/pip/wheels/97/a9/12/41b13e8b44889ab05ec4dcc91f27da21634bacf2a0e87473b8
  Running setup.py bdist_wheel for bcolz ... done
  Stored in directory: /root/.cache/pip/wheels/c5/cc/1b/2cf1f88959af5d7f4d449b7fc6c9452d0ecbd86fd61a9ee376
  Running setup.py bdist_wheel for alembic ... done
  Stored in directory: /root/.cache/pip/wheels/1f/04/83/76023f7a4c14688c0b5c2682a96392cfdd3ee4449eaaa287ef
  Running setup.py bdist_wheel for intervaltree ... done
  Stored in directory: /root/.cache/pip/wheels/08/99/c0/5a5942f5b9567c59c14aac76f95a70bf11dccc71240b91ebf5
  Running setup.py bdist_wheel for lru-dict ... done
  Stored in directory: /root/.cache/pip/wheels/b7/ef/06/fbdd555907a7d438fb33e4c8675f771ff1cf41917284c51ebf
  Running setup.py bdist_wheel for empyrical ... done
  Stored in directory: /root/.cache/pip/wheels/10/a4/3b/951bd609878a82fd72b9ea23699daf1eaada4ff6f583152876
  Running setup.py bdist_wheel for dill ... done
  Stored in directory: /root/.cache/pip/wheels/59/b1/91/f02e76c732915c4015ab4010f3015469866c1eb9b14058d8e7
  Running setup.py bdist_wheel for requests-ftp ... done
  Stored in directory: /root/.cache/pip/wheels/2a/98/32/37195e45a3392a73d9f65c488cbea30fe5bad76aaef4d6b020
Successfully built cvxpy plotly zipline scs multiprocess Logbook cyordereddict bottleneck bcolz alembic inter
valtree lru-dict empyrical dill requests-ftp
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.
zipline 1.2.0 has requirement pandas<0.19,>=0.18.1, but you'll have pandas 0.21.1 which is incompatible.
Installing collected packages: numpy, scipy, osqp, ecos, scs, dill, multiprocess, cvxpy, pandas, plotly, tqd
m, Logbook, requests-file, requests-ftp, pandas-datareader, cyordereddict, bottleneck, contextlib2, bcolz, mu
ltipledispatch, python-editor, alembic, sortedcontainers, intervaltree, lru-dict, empyrical, tables, zipline
  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 Logbook-1.5.3 alembic-1.4.2 bcolz-0.12.1 bottleneck-1.3.2 contextlib2-0.6.0.post1 cvxp
y-1.0.3 cyordereddict-1.0.0 dill-0.3.1.1 ecos-2.0.7.post1 empyrical-0.5.3 intervaltree-3.0.2 lru-dict-1.1.6 m
ultipledispatch-0.6.0 multiprocess-0.70.9 numpy-1.13.3 osqp-0.6.1 pandas-0.21.1 pandas-datareader-0.5.0 plotl
y-2.2.3 python-editor-1.0.4 requests-file-1.4.3 requests-ftp-0.3.1 scipy-1.0.0 scs-2.1.2 sortedcontainers-2.
1.0 tables-3.6.1 tqdm-4.19.5 zipline-1.2.0
```

Load Packages

In [1]:
```
#widening the display
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [2]:
```
import pandas as pd
import numpy as np
import helper
import project_helper
import project_tests
```

Market Data

Load Data

While using real data will give you hands on experience, it's doesn't cover all the topics we try to condense in one project. We'll solve this by creating new stocks. We've create a scenario where companies mining Terbium (https://en.wikipedia.org/wiki/Terbium) are making huge profits. All the companies in this sector of the market are made up. They represent a sector with large growth that will be used for demonstration latter in this project.

In [3]:
```
df_original = pd.read_csv('../../data/project_2/eod-quotemedia.csv', parse_dates=['date'], index_col=False)

# Add TB sector to the market
df = df_original
df = pd.concat([df] + project_helper.generate_tb_sector(df[df['ticker'] == 'AAPL']['date']), ignore_index=Tru
e)

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

print('Loaded Data')
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: FutureWarning:

Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.



Loaded Data
```

View Data

To see what one of these 2-d matrices looks like, let's take a look at the closing prices matrix.

In [4]:
```
close
```

Out[4]:

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	39.9
2013-07-02	29.65013670	15.81983388	80.72207258	54.31224742	35.42807578	50.69676639	31.27288084	64.71204071	46.03000000	39.8
2013-07-03	29.70518453	16.12794994	81.23729877	54.61204262	35.44486235	50.93716689	30.72565028	65.21451912	46.42000000	40.1
2013-07-05	30.43456826	16.21460758	81.82188233	54.17338125	35.85613355	51.37173702	31.32670680	66.07591068	47.00000000	40.6
2013-07-08	30.52402098	16.31089385	82.95141667	53.86579916	36.66188936	52.03746147	31.76628544	66.82065546	46.62500000	40.2
2013-07-09	30.68916447	16.71529618	82.43619048	54.81320389	36.35973093	51.69535307	31.16522893	66.48866080	47.26000000	40.6
2013-07-10	31.17771395	16.53235227	81.99032166	54.60295791	36.85493502	52.28710814	31.16522893	66.71298151	47.25000000	41.1
2013-07-11	31.45983407	16.72492481	82.00022986	55.45406479	37.08155384	53.72026495	31.85599537	67.47567196	47.99000000	42.2
2013-07-12	31.48047700	16.90786872	81.91105609	55.35309481	38.15724076	53.98840397	31.81096287	67.76280247	48.39000000	42.5
2013-07-15	31.72819223	17.10044125	82.61453801	55.47379158	37.79303181	53.84971137	31.95506689	68.41781897	48.12000000	42.5
2013-07-16	31.59057266	17.28338516	81.62371841	55.83133953	37.10696377	53.88669607	32.15320992	67.55642741	47.48500000	42.6
2013-07-17	31.38414330	17.76481650	80.74188897	55.84626440	37.23401341	54.06237335	32.26128793	67.43978064	48.04000000	42.8
2013-07-18	31.58369168	17.73593062	81.74261676	56.03418797	37.53893253	53.91443458	32.15320992	67.69101984	48.19000000	42.5
2013-07-19	31.79012104	17.55298671	81.45527908	55.15063572	37.70833205	54.37674323	32.30632044	67.49361761	48.07000000	42.2
2013-07-22	32.20297975	17.47595770	81.99032166	55.32713852	38.08948096	54.54317435	32.24327493	67.29621538	48.28000000	42.1
2013-07-23	31.97590746	17.37967143	81.94078068	54.37713815	37.53046256	53.28569482	33.03584705	66.62325323	48.07000000	42.5

ticker date	A	AAL	AAP	AAPL	ABBV	ABC	ABT	ACN	ADBE	
2013-07-24	32.17545584	17.81295964	80.78152175	57.17003539	36.96297418	52.49052395	32.82869752	66.14769330	47.80000000	42.4
2013-07-25	32.10664605	18.13070432	81.46518728	56.90917464	37.47117273	53.26720248	32.94578204	65.62726924	47.79000000	42.8
2013-07-26	31.37726233	18.38104862	81.88133151	57.23233050	37.93702140	54.06237335	33.12591207	65.60932358	47.64000000	42.7
2013-07-29	31.19835688	18.51584940	81.57417743	58.11484449	38.16571074	53.98840397	33.08988606	64.93636143	47.17000000	42.6
2013-07-30	30.86118893	18.48696352	81.43546269	58.83253602	37.86079161	53.84046520	33.21597708	66.16563896	47.36000000	43.0
2013-07-31	30.77861719	18.63139292	81.73270857	58.73000866	38.52144972	53.87744989	32.99081455	66.22844876	47.28000000	43.4
2013-08-01	31.68002538	18.66027880	82.66407899	59.26808264	38.32664028	54.36749706	33.17995107	67.16162295	47.70000000	43.9
2013-08-02	31.91397865	18.21736196	82.70371177	60.02912118	38.38593011	54.07161953	33.09889256	66.92832940	47.45000000	43.8
2013-08-05	31.61121560	18.45807764	82.64426260	60.92591114	37.86926159	54.58015904	32.82869752	66.60530757	47.63000000	43.6
2013-08-06	31.70754930	18.21736196	82.41637409	60.38082896	38.01325118	54.23805064	32.51346997	65.72597036	47.39000000	43.4
2013-08-07	31.84516887	18.16921883	81.53454465	60.34578796	37.75068193	54.31202002	32.36035945	65.50164964	47.10000000	43.2
2013-08-08	31.54928679	18.27513373	80.90042011	60.22638901	38.16571074	55.11643707	32.35135295	65.48370398	47.51000000	43.1
2013-08-09	31.80388300	17.90924591	82.40646589	59.36939001	37.86926159	55.00548300	32.32433344	65.97720956	47.18000000	43.1
2013-08-12	31.96214550	18.12107570	81.69307578	61.05595360	38.14877079	54.24729681	32.33333995	65.31322023	47.20000000	43.4
...	...	...	...	...	...	...	...	...	...	
2017-05-19	55.50327007	44.83282860	151.06072036	150.70113045	63.42995100	87.59994036	42.31486674	118.75601310	136.43000000	79.1
2017-05-22	55.45382835	45.81435227	146.97179877	151.61679784	63.29454092	87.91434122	42.87370534	120.45421034	138.86000000	79.9

ticker date	A	AAL	AAP	AAPL	ABBV	ABC	ABT	ACN	ADBE	
2017-05-23	58.00502088	46.26049939	140.27992953	151.42972601	63.68142686	87.62941544	42.82468441	119.90450488	139.52000000	79.8
2017-05-24	58.56865644	46.36955757	132.66057320	150.97681525	63.76847620	88.39576754	42.67762162	119.70818149	141.12000000	79.9
2017-05-25	58.63787484	47.60885514	131.61341035	151.49864721	64.14569000	89.51582062	43.08939743	120.83704093	142.85000000	80.2
2017-05-26	58.84553005	48.32269053	133.79749286	151.24265417	63.89421413	89.40774532	43.83451556	120.63090138	141.89000000	80.6
2017-05-30	59.69592756	47.54936885	132.63065426	151.30172949	63.85552554	89.43722040	44.11883696	121.49472426	142.41000000	82.6
2017-05-31	59.66626253	47.99551598	133.26892494	150.40575387	63.85552554	90.16427240	44.76591323	122.18185609	141.86000000	83.5
2017-06-01	60.05190791	48.63003633	136.72954883	150.81928108	64.52290380	91.51030109	45.19729742	122.98678196	141.38000000	80.0
2017-06-02	60.13101466	49.09601221	137.42765739	153.05429719	65.04519983	91.94260228	45.58946486	123.42850956	143.48000000	78.8
2017-06-05	59.72559259	49.31412858	135.20368297	151.55772252	65.29667569	91.68715157	45.70711509	124.25306776	143.59000000	76.7
2017-06-06	59.42894229	49.31412858	130.94522072	152.06970859	65.64487304	90.08567218	45.45220625	124.00766354	143.03000000	78.0
2017-06-07	59.95302448	50.42453920	130.26705812	152.97553010	66.49602213	90.32147283	45.64828997	124.22361925	143.62000000	79.1
2017-06-08	59.47838401	50.98965889	125.57975776	152.60138644	66.50569428	89.96777186	45.80515695	123.85060483	142.63000000	80.7
2017-06-09	58.54887976	49.83959075	128.01316475	146.68400898	67.38585980	90.48849829	46.36399555	123.50703891	138.05000000	76.9
2017-06-12	58.33133621	49.05635469	130.59616644	143.17887358	67.25044972	90.74394899	46.24634532	123.97821503	137.25000000	78.1
2017-06-13	58.61809816	49.02661155	131.27432905	144.33084223	67.38585980	91.37275071	46.53066671	124.73406004	139.09000000	79.5
2017-06-14	58.71698159	48.96712526	130.23713918	142.92288054	68.20799244	92.25700314	46.70714206	124.91075109	138.25000000	79.2
2017-06-15	58.54887976	48.68952261	130.79562603	142.06628846	68.28536963	92.77772957	47.17774299	124.69479537	137.52000000	78.1

ticker	A	AAL	AAP	AAPL	ABBV	ABC	ABT	ACN	ADBE	
date										
2017-06-16	58.84553005	48.37226243	129.80830106	140.07741950	68.72061632	90.91097444	47.26598066	125.21505233	137.84000000	78.4
2017-06-19	59.87391774	49.23481354	129.24981421	144.08469508	69.00110863	92.10962773	47.93266531	125.42119188	140.35000000	78.7
2017-06-20	59.65637419	47.61876952	123.24608056	142.77519225	68.88504285	91.61837639	47.81501508	124.20398692	140.91000000	77.5
2017-06-21	59.12240366	48.01534474	119.84529455	143.62193844	69.00110863	93.94690777	47.61893136	124.77332472	144.24000000	78.3
2017-06-22	59.93324780	48.55072128	120.43399427	143.38563718	70.78078399	94.68378480	48.30522438	119.83579169	143.69000000	79.6
2017-06-23	59.10262697	48.21363235	119.47610998	144.02561977	70.25848796	94.14340831	48.11894484	120.48365885	145.41000000	79.8
2017-06-26	58.57854478	48.36234805	121.52159207	143.57270901	70.35520945	94.31043377	47.95227368	120.09101209	144.96000000	78.9
2017-06-27	58.22256443	48.08474540	121.69121741	141.51491885	70.01668424	93.85848253	47.71697322	119.94376955	142.54000000	76.5
2017-06-28	58.73675827	48.82832394	116.45278767	143.58255490	70.52930812	94.69360982	47.53069368	121.46527575	143.81000000	77.5
2017-06-29	58.27398382	49.19515602	115.79424221	141.46568942	70.10373358	94.08445815	47.77579833	120.72906307	141.24000000	76.1
2017-06-30	58.77942143	49.88916265	116.33305213	141.80044954	70.13275003	92.87597984	47.65814810	121.40637874	141.44000000	76.2

1009 rows × 519 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 [5]:
```
apple_ticker = 'AAPL'
project_helper.plot_stock(close[apple_ticker], '{} Stock'.format(apple_ticker))
```

The Alpha Research Process

In this project you will code and evaluate a "breakout" signal. It is important to understand where these steps fit in the alpha research workflow. The signal-to-noise ratio in trading signals is very low and, as such, it is very easy to fall into the trap of *overfitting* to noise. It is therefore inadvisable to jump right into signal coding. To help mitigate overfitting, it is best to start with a general observation and hypothesis; i.e., you should be able to answer the following question *before* you touch any data:

> What feature of markets or investor behaviour would lead to a persistent anomaly that my signal will try to use?

Ideally the assumptions behind the hypothesis will be testable *before* you actually code and evaluate the signal itself. The workflow therefore is as follows:

 image

In this project, we assume that the first three steps area done ("observe & research", "form hypothesis", "validate hypothesis"). The hypothesis you'll be using for this project is the following:

- In the absence of news or significant investor trading interest, stocks oscillate in a range.
- Traders seek to capitalize on this range-bound behaviour periodically by selling/shorting at the top of the range and buying/covering at the bottom of the range. This behaviour reinforces the existence of the range.
- When stocks break out of the range, due to, e.g., a significant news release or from market pressure from a large investor:
  - the liquidity traders who have been providing liquidity at the bounds of the range seek to cover their positions to mitigate losses, thus magnifying the move out of the range, *and*
  - the move out of the range attracts other investor interest; these investors, due to the behavioural bias of *herding* (e.g., Herd Behavior (https://www.investopedia.com/university/behavioral_finance/behavioral8.asp)) build positions which favor continuation of the trend.

Using this hypothesis, let start coding..

Compute the Highs and Lows in a Window

You'll use the price highs and lows as an indicator for the breakout strategy. In this section, implement `get_high_lows_lookback` to get the maximum high price and minimum low price over a window of days. The variable `lookback_days` contains the number of days to look in the past. Make sure this doesn't include the current day.

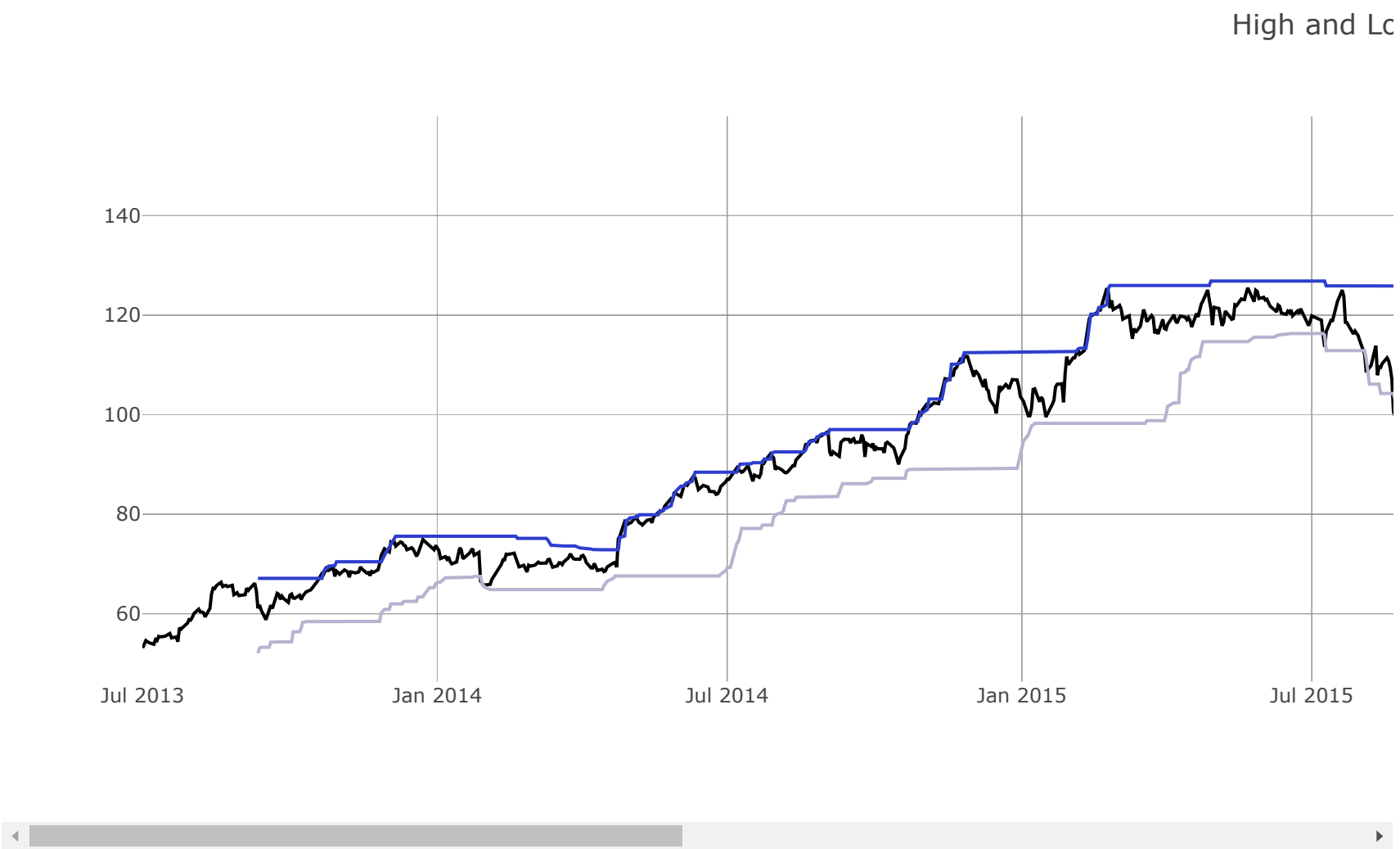
In [6]:
```
def get_high_lows_lookback(high, low, lookback_days):
    """
    Get the highs and lows in a lookback window.

    Parameters
    ----------
    high : DataFrame
        High price for each ticker and date
    low : DataFrame
        Low price for each ticker and date
    lookback_days : int
        The number of days to look back

    Returns
    -------
    lookback_high : DataFrame
        Lookback high price for each ticker and date
    lookback_low : DataFrame
        Lookback low price for each ticker and date
    """
    Max_High = high.shift(periods=1).rolling(window=lookback_days).max()
    Min_Low  =  low.shift(periods=1).rolling(window=lookback_days).min()

    return Max_High, Min_Low

project_tests.test_get_high_lows_lookback(get_high_lows_lookback)
```

```
Tests Passed
```

View Data

Let's use your implementation of `get_high_lows_lookback` to get the highs and lows for the past 50 days and compare it to it their respective stock. Just like last time, we'll use Apple's stock as the example to look at.

In [7]:
```
lookback_days = 50
lookback_high, lookback_low = get_high_lows_lookback(high, low, lookback_days)
project_helper.plot_high_low(
    close[apple_ticker],
    lookback_high[apple_ticker],
    lookback_low[apple_ticker],
    'High and Low of {} Stock'.format(apple_ticker))
```



Compute Long and Short Signals

Using the generated indicator of highs and lows, create long and short signals using a breakout strategy. Implement `get_long_short` to generate the following signals:

Signal	Condition
-1	Low > Close Price
1	High < Close Price
0	Otherwise

In this chart, **Close Price** is the `close` parameter. **Low** and **High** are the values generated from `get_high_lows_lookback`, the `lookback_high` and `lookback_low` parameters.

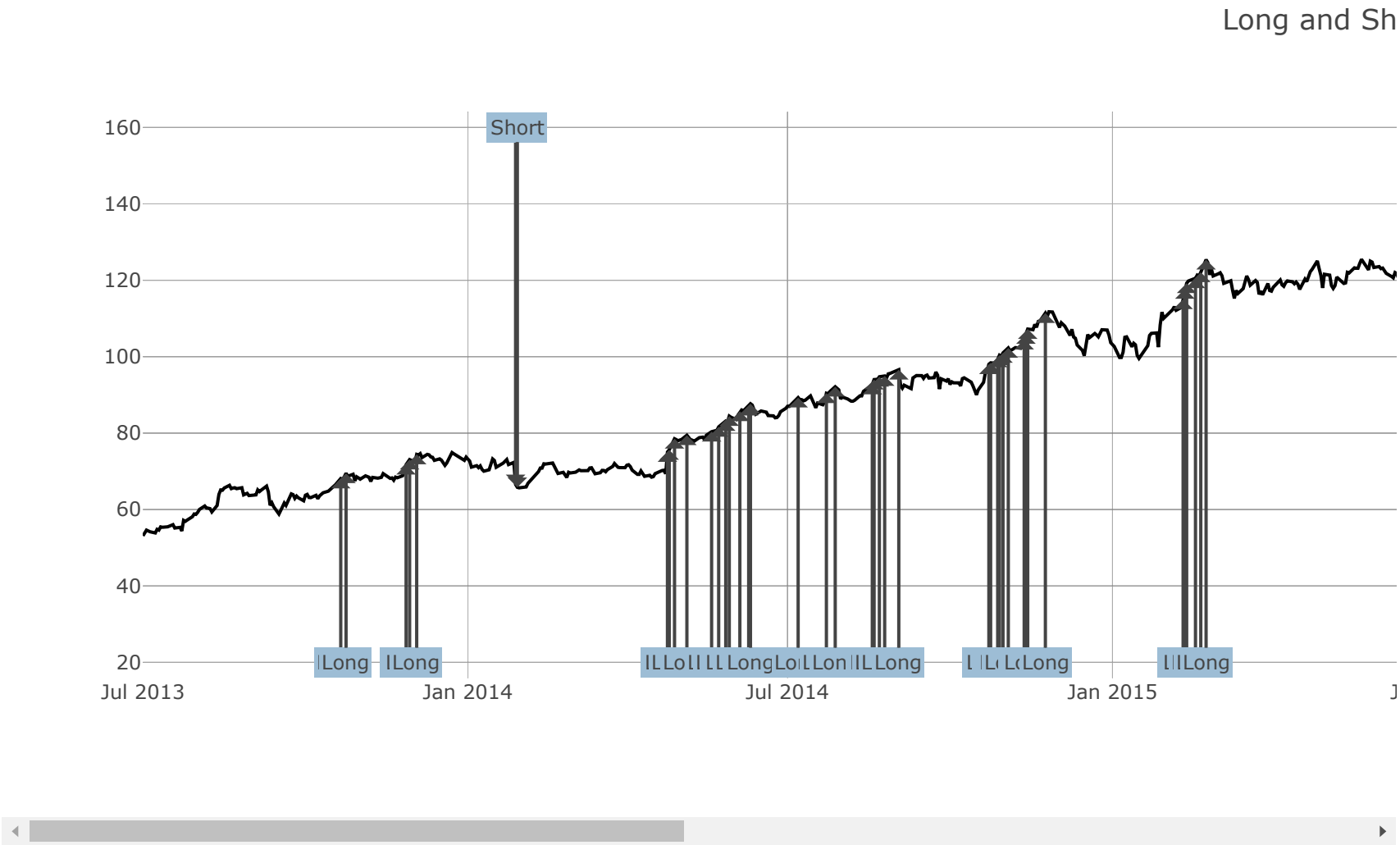
In [8]:
```
def get_long_short(close, lookback_high, lookback_low):
    """
    Generate the signals long, short, and do nothing.

    Parameters
    ----------
    close : DataFrame
        Close price for each ticker and date
    lookback_high : DataFrame
        Lookback high price for each ticker and date
    lookback_low : DataFrame
        Lookback low price for each ticker and date

    Returns
    -------
    long_short : DataFrame
        The long, short, and do nothing signals for each ticker and date
    """
    #TODO: Implement function
    signal = pd.DataFrame(0,index = close.index, columns = close.columns)
    signal[close > lookback_high] = 1
    signal[close < lookback_low] = -1
    return signal


project_tests.test_get_long_short(get_long_short)
```

```
Tests Passed
```

View Data

Let's compare the signals you generated against the close prices. This chart will show a lot of signals. Too many in fact. We'll talk about filtering the redundant signals in the next problem.

In [9]:
```
signal = get_long_short(close, lookback_high, lookback_low)
project_helper.plot_signal(
    close[apple_ticker],
    signal[apple_ticker],
    'Long and Short of {} Stock'.format(apple_ticker))
```



Filter Signals

That was a lot of repeated signals! If we're already shorting a stock, having an additional signal to short a stock isn't helpful for this strategy. This also applies to additional long signals when the last signal was long.

Implement `filter_signals` to filter out repeated long or short signals within the `lookahead_days`. If the previous signal was the same, change the signal to `0` (do nothing signal). For example, say you have a single stock time series that is

```
[1, 0, 1, 0, 1, 0, -1, -1]
```

Running `filter_signals` with a lookahead of 3 days should turn those signals into

```
[1, 0, 0, 0, 1, 0, -1, 0]
```

To help you implement the function, we have provided you with the `clear_signals` function. This will remove all signals within a window after the last signal. For example, say you're using a windows size of 3 with `clear_signals`. It would turn the Series of long signals

```
[0, 1, 0, 0, 1, 1, 0, 1, 0]
```

into

```
[0, 1, 0, 0, 0, 1, 0, 0, 0]
```

`clear_signals` only takes a Series of the same type of signals, where `1` is the signal and `0` is no signal. It can't take a mix of long and short signals. Using this function, implement `filter_signals`.

For implementing `filter_signals`, we don't reccommend you try to find a vectorized solution. Instead, you should use the `iterrows` (https://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.DataFrame.iterrows.html) over each column.

In [10]:
```
def clear_signals(signals, window_size):
    """
    Clear out signals in a Series of just long or short signals.

    Remove the number of signals down to 1 within the window size time period.

    Parameters
    ----------
    signals : Pandas Series
        The long, short, or do nothing signals
    window_size : int
        The number of days to have a single signal

    Returns
    -------
    signals : Pandas Series
        Signals with the signals removed from the window size
    """
    # Start with buffer of window size
    # This handles the edge case of calculating past_signal in the beginning
    clean_signals = [0]*window_size

    for signal_i, current_signal in enumerate(signals):
        # Check if there was a signal in the past window_size of days
        has_past_signal = bool(sum(clean_signals[signal_i:signal_i+window_size]))
        # Use the current signal if there's no past signal, else 0/False
        clean_signals.append(not has_past_signal and current_signal)

    # Remove buffer
    clean_signals = clean_signals[window_size:]

    # Return the signals as a Series of Ints
    return pd.Series(np.array(clean_signals).astype(np.int), signals.index)


def filter_signals(signal, lookahead_days):
    """
    Filter out signals in a DataFrame.

    Parameters
    ----------
    signal : DataFrame
```

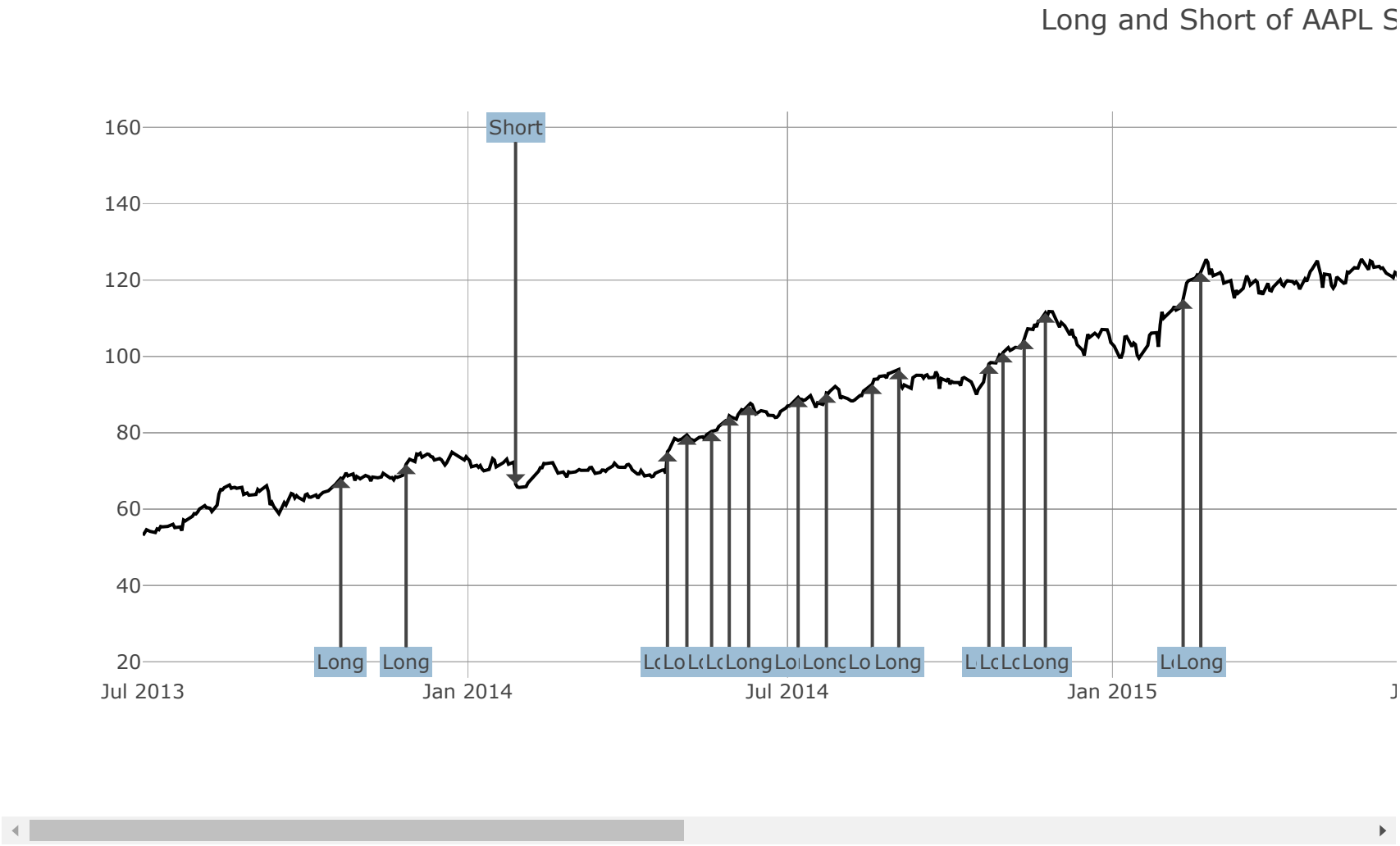
```
        The long, short, and do nothing signals for each ticker and date
    lookahead_days : int
        The number of days to look ahead

    Returns
    -------
    filtered_signal : DataFrame
        The filtered long, short, and do nothing signals for each ticker and date
    """
    #TODO: Implement function
    positive_signals = signal.where(signal==1, 0)
    negative_signals = signal.where(signal==-1, 0)

    for col in signal.columns:
        positive_signals[col] = clear_signals(positive_signals[col], lookahead_days)
        negative_signals[col] = clear_signals(negative_signals[col], lookahead_days)

    filtered_signal = positive_signals + negative_signals

    return filtered_signal
    return None

project_tests.test_filter_signals(filter_signals)
```

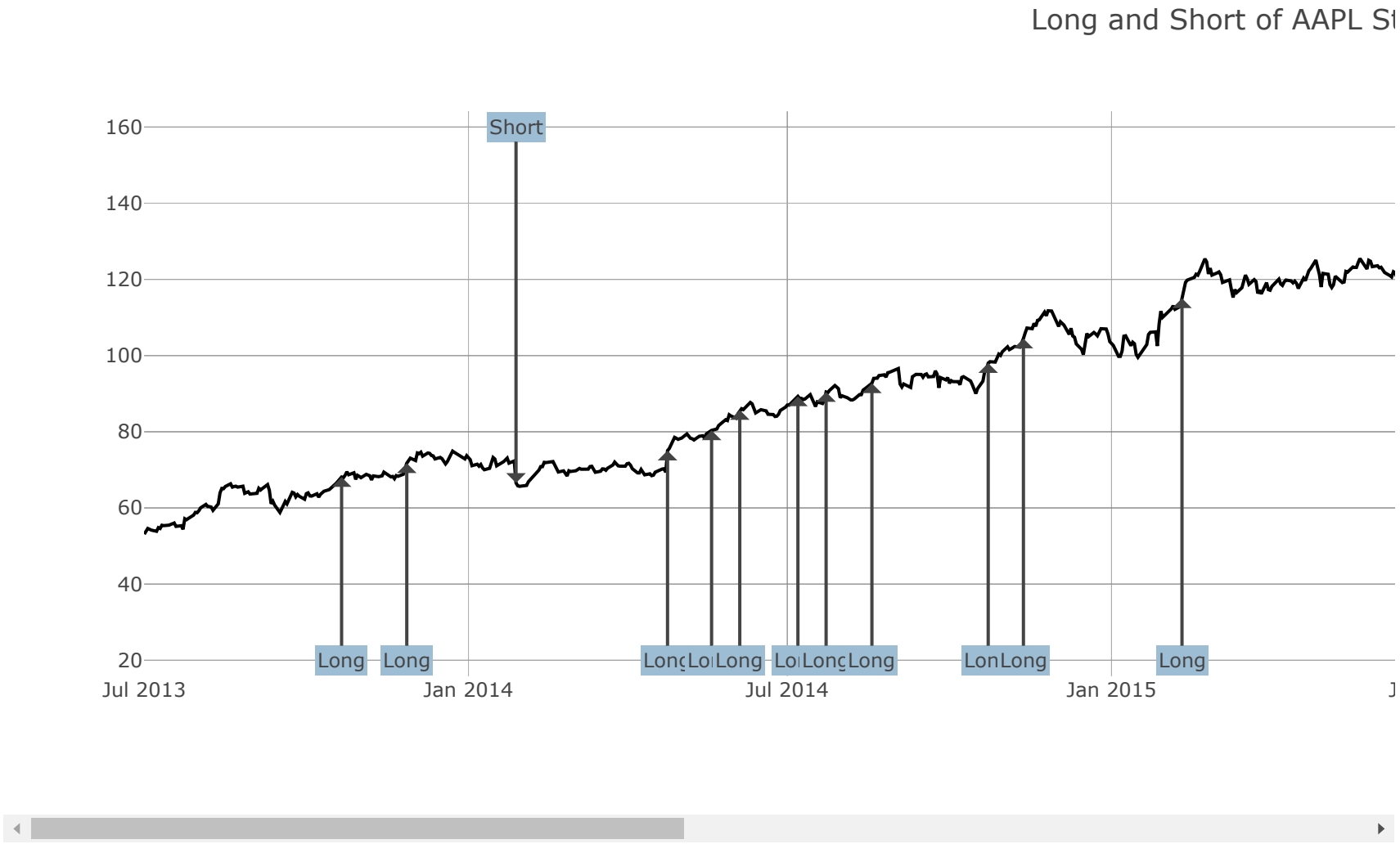
```
Tests Passed
```

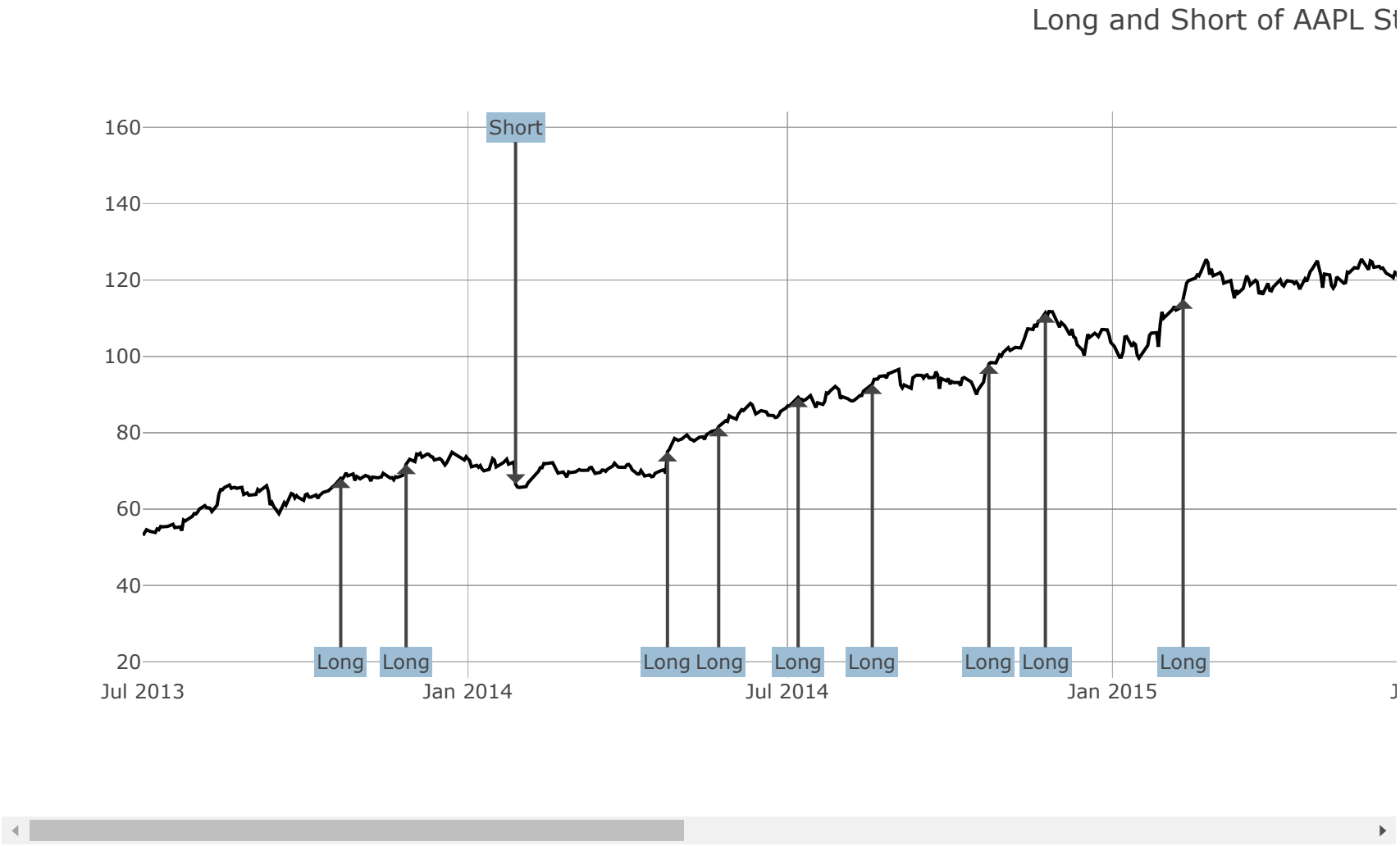
View Data

Let's view the same chart as before, but with the redundant signals removed.

In [11]:
```
signal_5 = filter_signals(signal, 5)
signal_10 = filter_signals(signal, 10)
signal_20 = filter_signals(signal, 20)
for signal_data, signal_days in [(signal_5, 5), (signal_10, 10), (signal_20, 20)]:
    project_helper.plot_signal(
        close[apple_ticker],
        signal_data[apple_ticker],
        'Long and Short of {} Stock with {} day signal window'.format(apple_ticker, signal_days))
```


Long and Short of AAPL S
160
140
120
100
80
60
40
20
Short
Long
Long
Jul 2013
Jan 2014
Jul 2014
Jan 2015


Long and Short of AAPL St
160
140
120
100
80
60
40
20
Short
Long
Long
Long
Long
Long
Long
Long
Long
Long
Long
Long
Jul 2013
Jan 2014
Jul 2014
Jan 2015



Lookahead Close Prices

With the trading signal done, we can start working on evaluating how many days to short or long the stocks. In this problem, implement `get_lookahead_prices` to get the close price days ahead in time. You can get the number of days from the variable `lookahead_days`. We'll use the lookahead prices to calculate future returns in another problem.

In [12]:
```
def get_lookahead_prices(close, lookahead_days):
    """
    Get the lookahead prices for `lookahead_days` number of days.

    Parameters
    ----------
    close : DataFrame
        Close price for each ticker and date
    lookahead_days : int
        The number of days to look ahead

    Returns
    -------
    lookahead_prices : DataFrame
        The lookahead prices for each ticker and date
    """
    #TODO: Implement function
    return close.shift(-1 * lookahead_days)

project_tests.test_get_lookahead_prices(get_lookahead_prices)
```

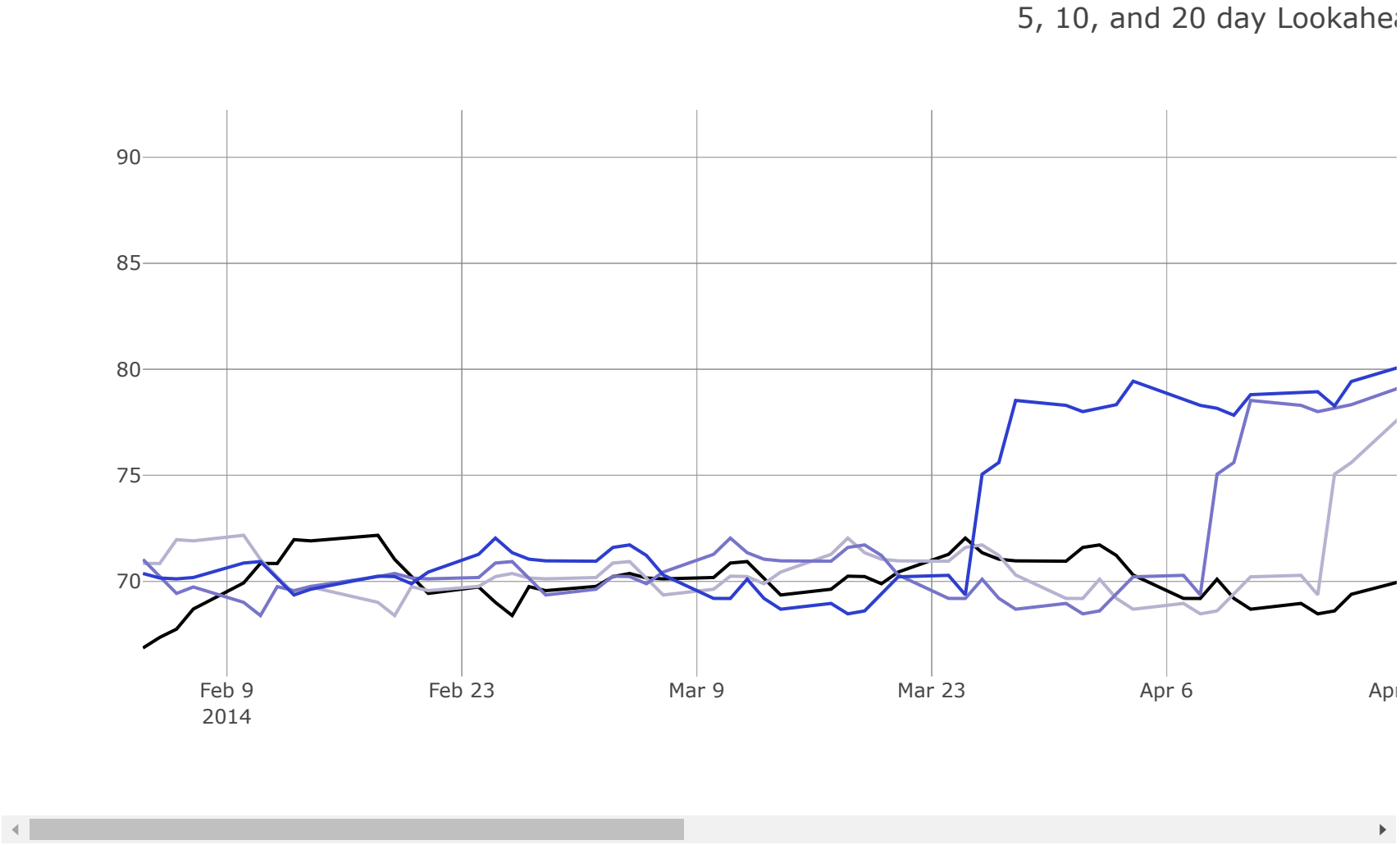
```
Tests Passed
```

View Data

Using the `get_lookahead_prices` function, let's generate lookahead closing prices for 5, 10, and 20 days.

Let's also chart a subsection of a few months of the Apple stock instead of years. This will allow you to view the differences between the 5, 10, and 20 day lookaheads. Otherwise, they will mesh together when looking at a chart that is zoomed out.

In [13]:
```
lookahead_5 = get_lookahead_prices(close, 5)
lookahead_10 = get_lookahead_prices(close, 10)
lookahead_20 = get_lookahead_prices(close, 20)
project_helper.plot_lookahead_prices(
    close[apple_ticker].iloc[150:250],
    [
        (lookahead_5[apple_ticker].iloc[150:250], 5),
        (lookahead_10[apple_ticker].iloc[150:250], 10),
        (lookahead_20[apple_ticker].iloc[150:250], 20)],
    '5, 10, and 20 day Lookahead Prices for Slice of {} Stock'.format(apple_ticker))
```



Lookahead Price Returns

Implement `get_return_lookahead` to generate the log price return between the closing price and the lookahead price.

In [14]:
```
def get_return_lookahead(close, lookahead_prices):
    """
    Calculate the log returns from the lookahead days to the signal day.

    Parameters
    ----------
    close : DataFrame
        Close price for each ticker and date
    lookahead_prices : DataFrame
        The lookahead prices for each ticker and date

    Returns
    -------
    lookahead_returns : DataFrame
        The lookahead log returns for each ticker and date
    """
    #TODO: Implement function

    return np.log(lookahead_prices) - np.log(close)

project_tests.test_get_return_lookahead(get_return_lookahead)
```

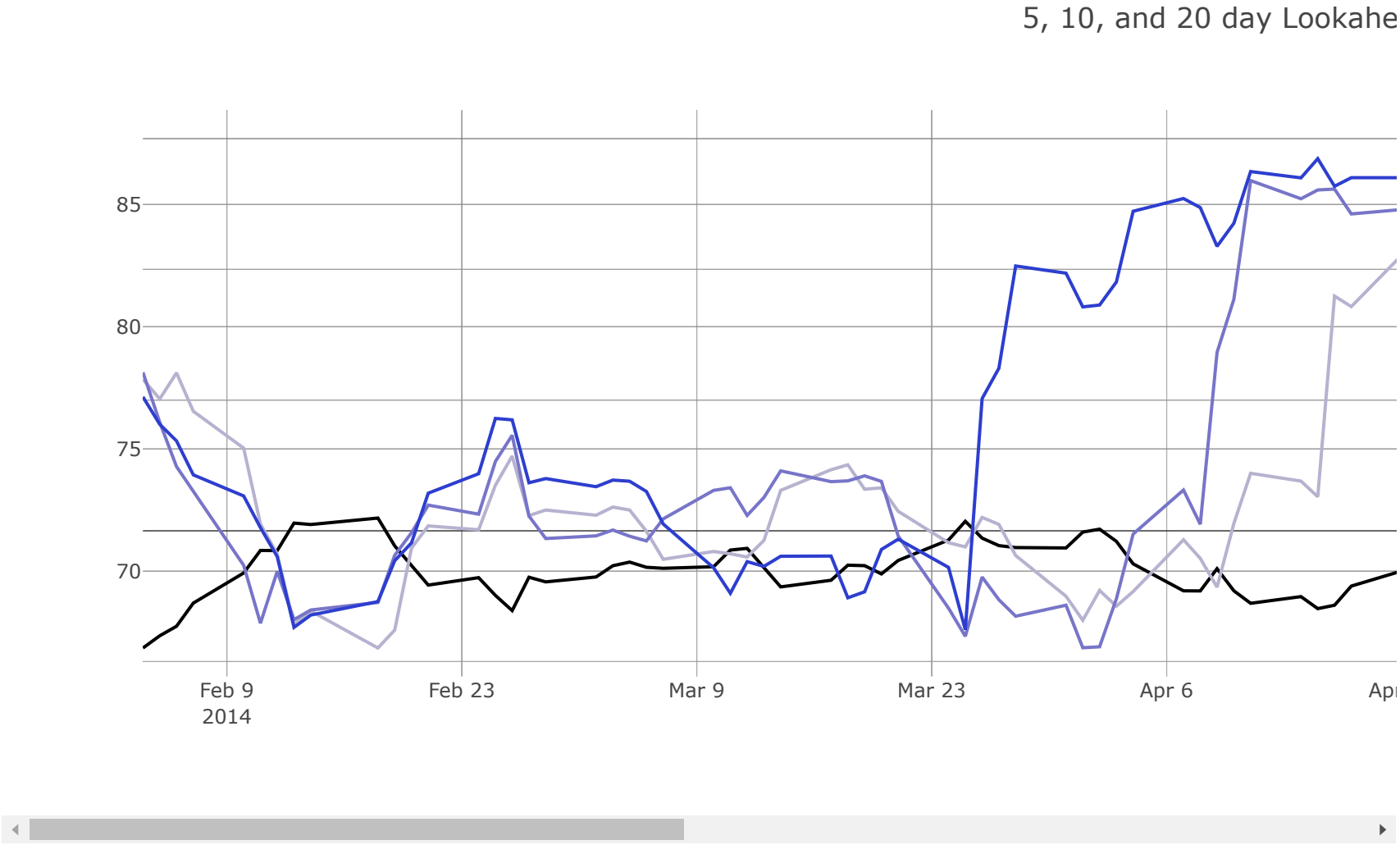
```
Tests Passed
```

View Data

Using the same lookahead prices and same subsection of the Apple stock from the previous problem, we'll view the lookahead returns.

In order to view price returns on the same chart as the stock, a second y-axis will be added. When viewing this chart, the axis for the price of the stock will be on the left side, like previous charts. The axis for price returns will be located on the right side.

In [15]:
```
price_return_5 = get_return_lookahead(close, lookahead_5)
price_return_10 = get_return_lookahead(close, lookahead_10)
price_return_20 = get_return_lookahead(close, lookahead_20)
project_helper.plot_price_returns(
    close[apple_ticker].iloc[150:250],
    [
        (price_return_5[apple_ticker].iloc[150:250], 5),
        (price_return_10[apple_ticker].iloc[150:250], 10),
        (price_return_20[apple_ticker].iloc[150:250], 20)],
    '5, 10, and 20 day Lookahead Returns for Slice {} Stock'.format(apple_ticker))
```



Compute the Signal Return

Using the price returns generate the signal returns.

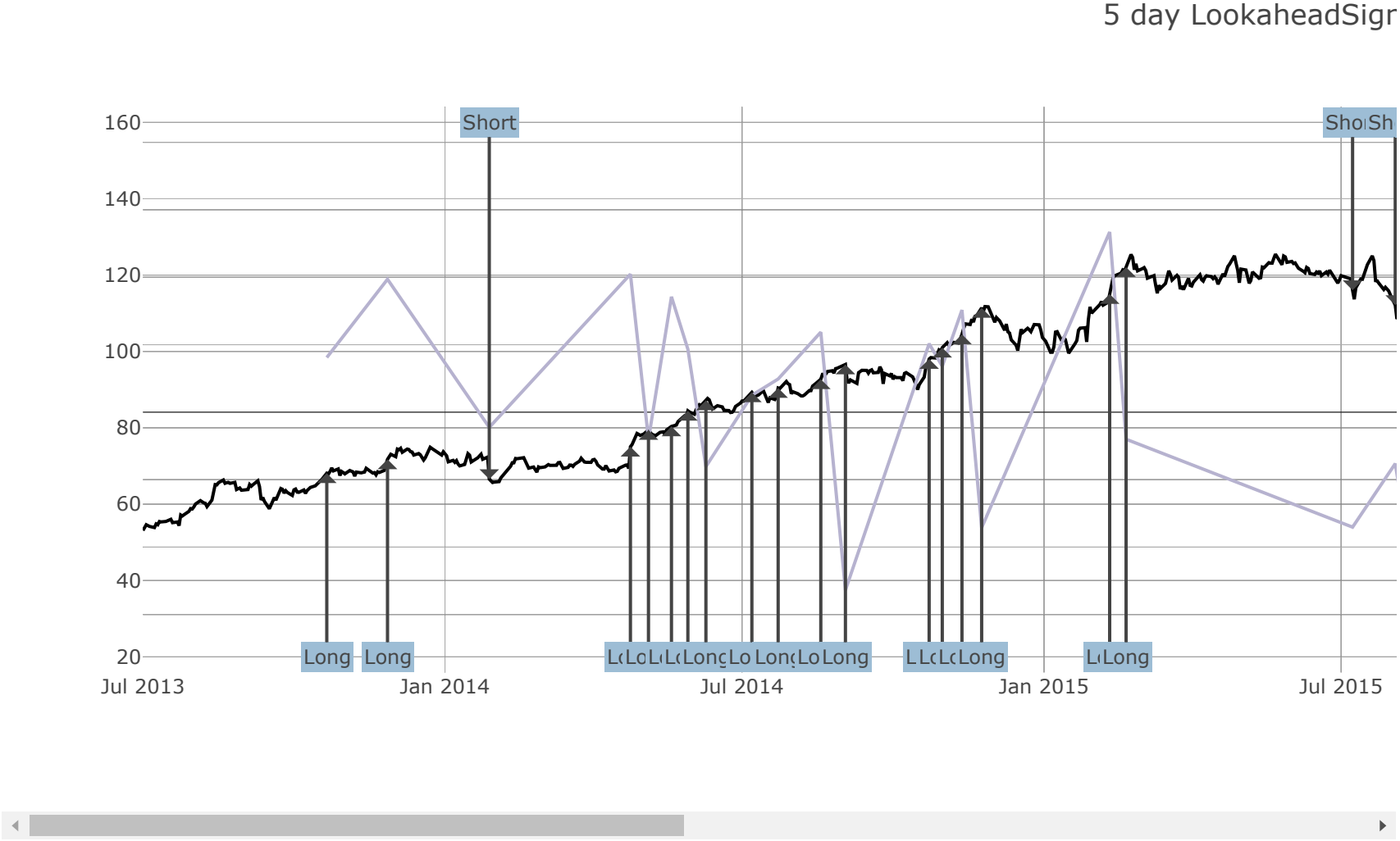
In [16]:
```
def get_signal_return(signal, lookahead_returns):
    """
    Compute the signal returns.

    Parameters
    ----------
    signal : DataFrame
        The long, short, and do nothing signals for each ticker and date
    lookahead_returns : DataFrame
        The lookahead log returns for each ticker and date

    Returns
    -------
    signal_return : DataFrame
        Signal returns for each ticker and date
    """
    #TODO: Implement function

    return signal*lookahead_returns

project_tests.test_get_signal_return(get_signal_return)
```

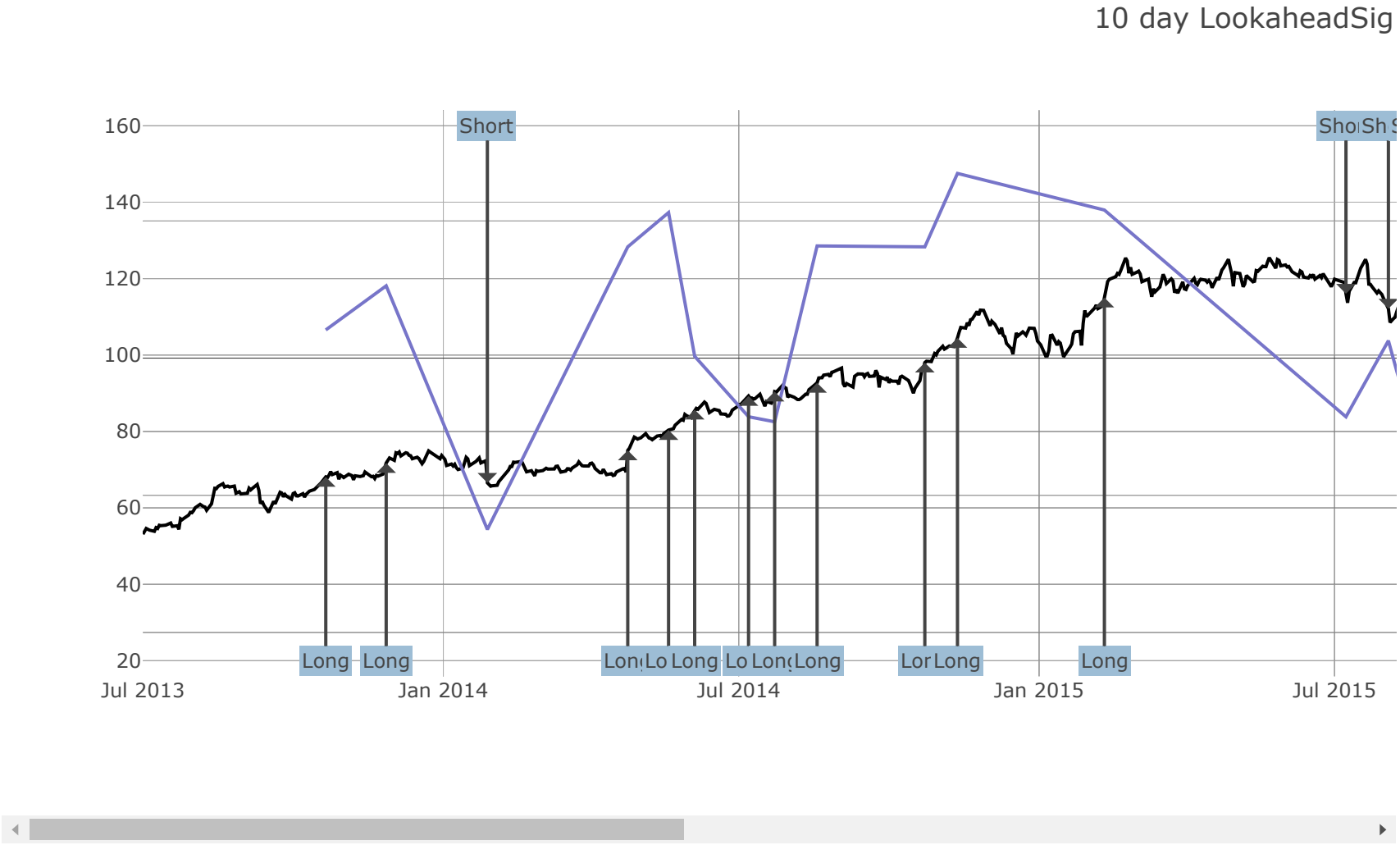
```
Tests Passed
```

View Data

Let's continue using the previous lookahead prices to view the signal returns. Just like before, the axis for the signal returns is on the right side of the chart.

In [17]:
```
title_string = '{} day LookaheadSignal Returns for {} Stock'
signal_return_5 = get_signal_return(signal_5, price_return_5)
signal_return_10 = get_signal_return(signal_10, price_return_10)
signal_return_20 = get_signal_return(signal_20, price_return_20)
project_helper.plot_signal_returns(
    close[apple_ticker],
    [
        (signal_return_5[apple_ticker], signal_5[apple_ticker], 5),
        (signal_return_10[apple_ticker], signal_10[apple_ticker], 10),
        (signal_return_20[apple_ticker], signal_20[apple_ticker], 20)],
    [title_string.format(5, apple_ticker), title_string.format(10, apple_ticker), title_string.format(20, app
le_ticker)])
```


5 day LookaheadSigr
160
140
120
100
80
60
40
20
Short
Sho
Sh
Long
Long
Long
Long
Long
Long
Long
Long
Long
Long
Long
Jul 2013
Jan 2014
Jul 2014
Jan 2015
Jul 2015

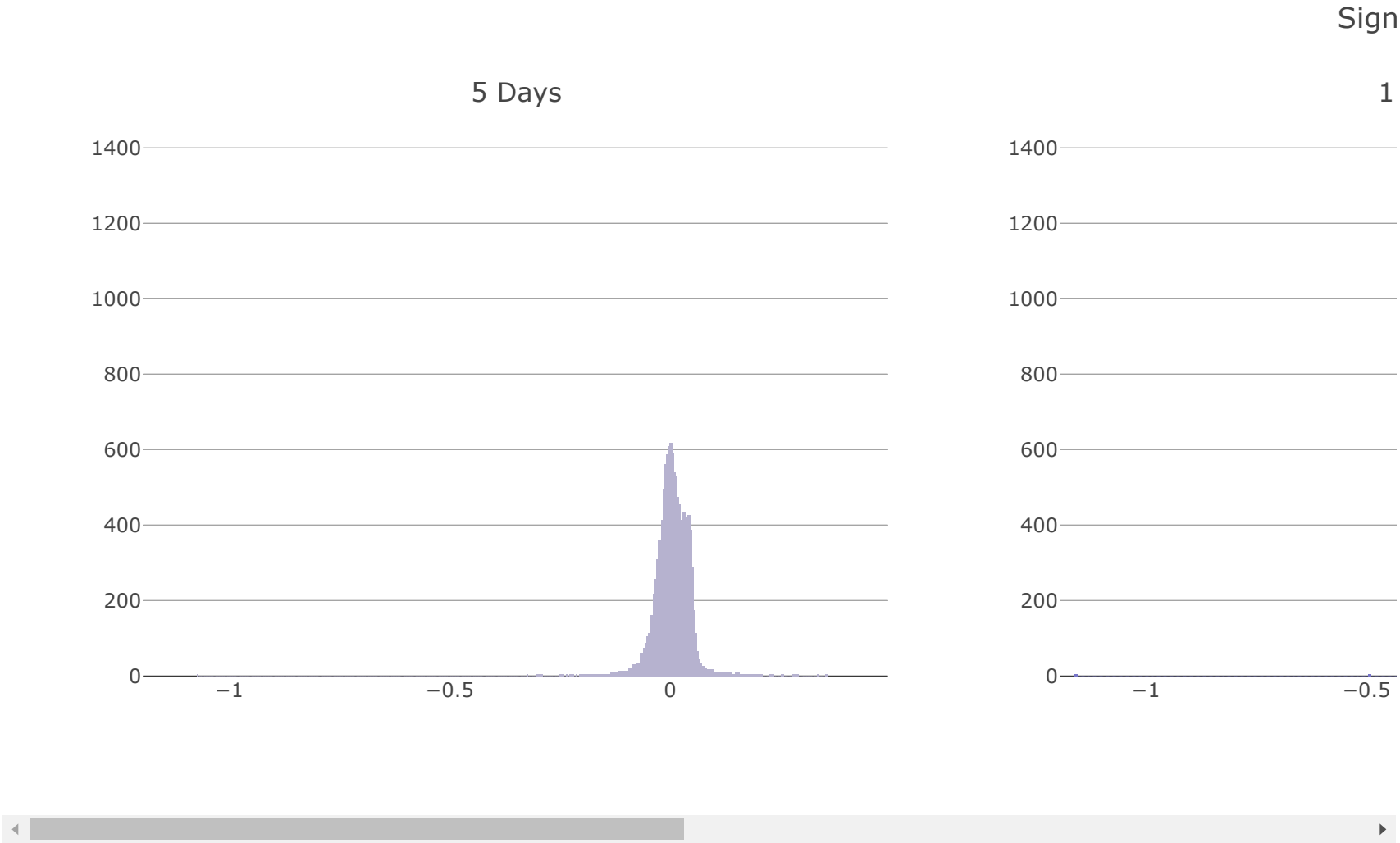

10 day LookaheadSig
160
140
120
100
80
60
40
20
Short
Long
Long
Long
Long
Long
Long
Long
Long
Long
Long
Jul 2013
Jan 2014
Jul 2014
Jan 2015
Jul 2015

Test for Significance

Histogram

Let's plot a histogram of the signal return values.

In [18]:
```
project_helper.plot_signal_histograms(
    [signal_return_5, signal_return_10, signal_return_20],
    'Signal Return',
    ('5 Days', '10 Days', '20 Days'))
```



Question: What do the histograms tell you about the signal returns?

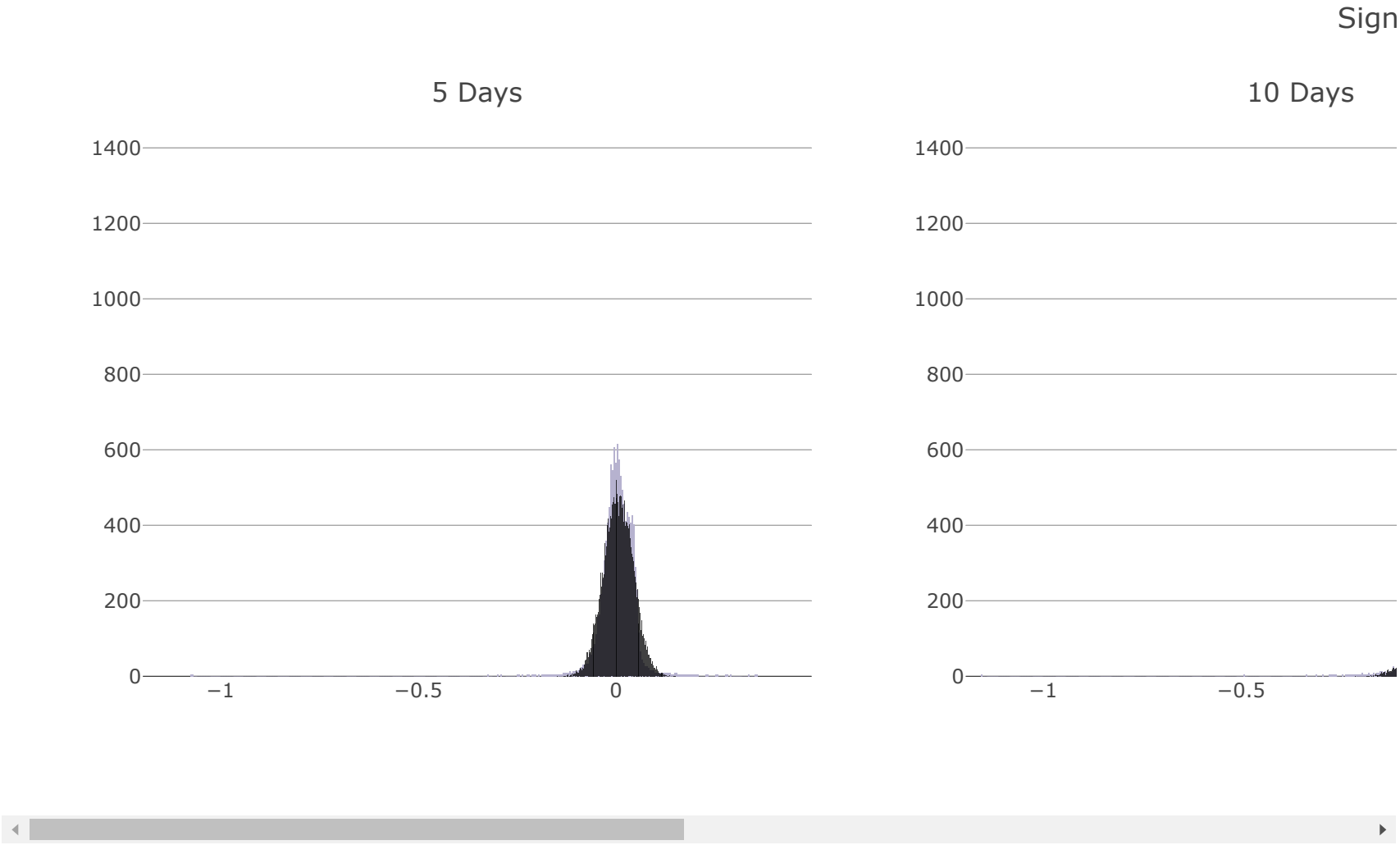
#TODO: Put Answer In this Cell

Obviously the returns is not normal distribution and is close to bimodal disturbution. This effect is more pronounced in higher lookahead period. Which reflects more companies giving larger returns.

Outliers

You might have noticed the outliers in the 10 and 20 day histograms. To better visualize the outliers, let's compare the 5, 10, and 20 day signals returns to normal distributions with the same mean and deviation for each signal return distributions.

In [19]:
```
project_helper.plot_signal_to_normal_histograms(
    [signal_return_5, signal_return_10, signal_return_20],
    'Signal Return',
    ('5 Days', '10 Days', '20 Days'))
```



Kolmogorov-Smirnov Test

While you can see the outliers in the histogram, we need to find the stocks that are causing these outlying returns. We'll use the Kolmogorov-Smirnov Test or KS-Test. This test will be applied to teach ticker's signal returns where a long or short signal exits.

In [20]:
```
# Filter out returns that don't have a long or short signal.
long_short_signal_returns_5 = signal_return_5[signal_5 != 0].stack()
long_short_signal_returns_10 = signal_return_10[signal_10 != 0].stack()
long_short_signal_returns_20 = signal_return_20[signal_20 != 0].stack()

# Get just ticker and signal return
long_short_signal_returns_5 = long_short_signal_returns_5.reset_index().iloc[:, [1,2]]
long_short_signal_returns_5.columns = ['ticker', 'signal_return']
long_short_signal_returns_10 = long_short_signal_returns_10.reset_index().iloc[:, [1,2]]
long_short_signal_returns_10.columns = ['ticker', 'signal_return']
long_short_signal_returns_20 = long_short_signal_returns_20.reset_index().iloc[:, [1,2]]
long_short_signal_returns_20.columns = ['ticker', 'signal_return']

# View some of the data
long_short_signal_returns_5.head(10)
```

Out[20]:

	ticker	signal_return
0	A	0.00732604
1	ABC	0.01639650
2	ADP	0.00981520
3	AGENEN	0.02155628
4	AKAM	0.04400495
5	ALGN	0.01545561
6	ALTAIC	0.01490812
7	APC	0.00305859
8	ARMENA	0.01858008
9	BA	0.08061297

This gives you the data to use in the KS-Test.

Now it's time to implement the function `calculate_kstest` to use Kolmogorov-Smirnov test (KS test) between a distribution of stock returns (the input dataframe in this case) and each stock's signal returns. Run KS test on a normal distribution against each stock's signal returns. Use `scipy.stats.kstest` (https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.kstest.html#scipy-stats-kstest) perform the KS test. When calculating the standard deviation of the signal returns, make sure to set the delta degrees of freedom to 0.

For this function, we don't reccommend you try to find a vectorized solution. Instead, you should iterate over the `groupby` (https://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.DataFrame.groupby.html) function.

In [21]:
```
from scipy.stats import kstest


def calculate_kstest(long_short_signal_returns):
    """
    Calculate the KS-Test against the signal returns with a long or short signal.

    Parameters
    ----------
    long_short_signal_returns : DataFrame
        The signal returns which have a signal.
        This DataFrame contains two columns, "ticker" and "signal_return"

    Returns
    -------
    ks_values : Pandas Series
        KS static for all the tickers
    p_values : Pandas Series
        P value for all the tickers
    """
    #TODO: Implement function
    ks_values = pd.Series()
    p_values = pd.Series()
    mean = long_short_signal_returns['signal_return'].mean()
    std = long_short_signal_returns['signal_return'].std(ddof=0)
    normal_dist_args = (mean, std)

    for name, group in long_short_signal_returns.groupby('ticker'):
        sample = group['signal_return']
        ks_value, p_value = kstest(sample, 'norm', normal_dist_args)
        ks_values[name] = ks_value
        p_values[name] = p_value

    return ks_values, p_values


project_tests.test_calculate_kstest(calculate_kstest)
```

```
Tests Passed
```

View Data

Using the signal returns we created above, let's calculate the ks and p values.

In [22]:
```
ks_values_5, p_values_5 = calculate_kstest(long_short_signal_returns_5)
ks_values_10, p_values_10 = calculate_kstest(long_short_signal_returns_10)
ks_values_20, p_values_20 = calculate_kstest(long_short_signal_returns_20)

print('ks_values_5')
print(ks_values_5.head(10))
print('p_values_5')
print(p_values_5.head(10))
```

```
ks_values_5
A      0.17204342
AAL    0.10739736
AAP    0.19687672
AAPL   0.15579823
ABBV   0.16807045
ABC    0.21425384
ABT    0.21363463
ACN    0.28211891
ADBE   0.24258364
ADI    0.19419650
dtype: float64
p_values_5
A      0.18759906
AAL    0.72499401
AAP    0.04517062
AAPL   0.24621414
ABBV   0.24740898
ABC    0.02718514
ABT    0.04846057
ACN    0.00587908
ADBE   0.00916318
ADI    0.09916274
dtype: float64
```

Find Outliers

With the ks and p values calculate, let's find which symbols are the outliers. Implement the `find_outliers` function to find the following outliers:

- Symbols that pass the null hypothesis with a p-value less than `pvalue_threshold` .
- Symbols that with a KS value above `ks_threshold` .

In [23]:
```
def find_outliers(ks_values, p_values, ks_threshold, pvalue_threshold=0.05):
    """
    Find outlying symbols using KS values and P-values

    Parameters
    ----------
    ks_values : Pandas Series
        KS static for all the tickers
    p_values : Pandas Series
        P value for all the tickers
    ks_threshold : float
        The threshold for the KS statistic
    pvalue_threshold : float
        The threshold for the p-value

    Returns
    -------
    outliers : set of str
        Symbols that are outliers
    """
    #TODO: Implement function
    set_=set(ks_values.index[(ks_values>ks_threshold)&(p_values<pvalue_threshold)])
    return set_


project_tests.test_find_outliers(find_outliers)
```

```
Tests Passed
```

View Data

Using the `find_outliers` function you implemented, let's see what we found.

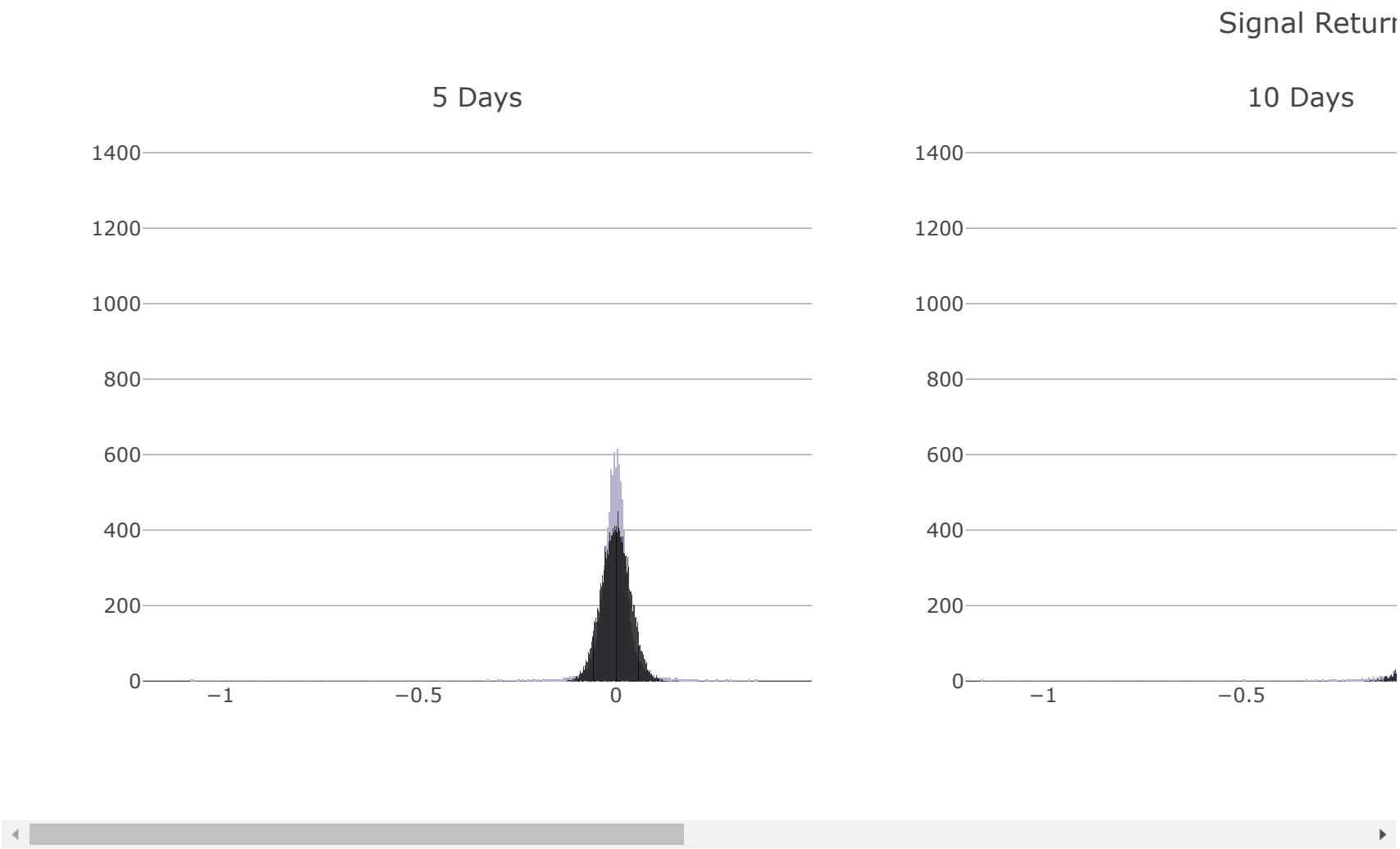
In [24]:
```
ks_threshold = 0.8
outliers_5 = find_outliers(ks_values_5, p_values_5, ks_threshold)
outliers_10 = find_outliers(ks_values_10, p_values_10, ks_threshold)
outliers_20 = find_outliers(ks_values_20, p_values_20, ks_threshold)

outlier_tickers = outliers_5.union(outliers_10).union(outliers_20)
print('{} Outliers Found:\n{}'.format(len(outlier_tickers), ', '.join(list(outlier_tickers))))
```

```
24 Outliers Found:
LINIFO, DASYST, SYLVES, TARDA, VVEDEN, PRAEST, HUMILI, URUMIE, BIFLOR, GREIGI, SPRENG, BAKERI, PULCHE, ALTAI
C, AGENEN, GESNER, SAXATI, ARMENA, CLUSIA, SCHREN, ORPHAN, KOLPAK, KAUFMA, TURKES
```

Show Significance without Outliers

Let's compare the 5, 10, and 20 day signals returns without outliers to normal distributions. Also, let's see how the P-Value has changed with the outliers removed.

In [25]:
```
good_tickers = list(set(close.columns) - outlier_tickers)

project_helper.plot_signal_to_normal_histograms(
    [signal_return_5[good_tickers], signal_return_10[good_tickers], signal_return_20[good_tickers]],
    'Signal Return Without Outliers',
    ('5 Days', '10 Days', '20 Days'))
```



That's more like it! The returns are closer to a normal distribution. You have finished the research phase of a Breakout Strategy. You can now submit your project.

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.