Project 8: Backtesting

In this project, you will build a fairly realistic backtester that uses the Barra data. The backtester will perform portfolio optimization that includes transaction costs, and you'll implement it with computational efficiency in mind, to allow for a reasonably fast backtest. You'll also use performance attribution to identify the major drivers of your portfolio's profit-and-loss (PnL). You will have the option to modify and customize the backtest as well.

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. 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.

Install Packages

In [1]:
```
import sys
!{sys.executable} -m pip install -r requirements.txt
```

```
Requirement already satisfied: matplotlib==2.1.0 in /opt/conda/lib/python
3.6/site-packages (from -r requirements.txt (line 1)) (2.1.0)
Collecting numpy==1.16.1 (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/f5/bf/4981bcbee4393
4f0adb8f764a1e70ab0ee5a448f6505bd04a87a2fda2a8b/numpy-1.16.1-cp36-cp36m-m
anylinux1_x86_64.whl (17.3MB)
    100% |████████████████████████████████| 17.3MB 2.0MB/s eta 0:00:01
3% |█                               | 542kB 5.0MB/s eta 0:00:04    28% |█
████████                       | 4.9MB 35.2MB/s eta 0:00:01    99% |█████
███████████████████████████| 17.2MB 30.0MB/s eta 0:00:01
Collecting pandas==0.24.1 (from -r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/e6/de/a0d3defd8f338
eaf53ef716e40ef6d6c277c35d50e09b586e170169cdf0d/pandas-0.24.1-cp36-cp36m-
manylinux1_x86_64.whl (10.1MB)
    100% |████████████████████████████████| 10.1MB 4.8MB/s eta 0:00:01
36% |████████████                    | 3.7MB 30.5MB/s eta 0:00:01
```

```
Collecting patsy==0.5.1 (from -r requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/ea/0c/5f61f1a3d4385
d6bf83b83ea495068857ff8dfb89e74824c6e9eb63286d8/patsy-0.5.1-py2.py3-none-
any.whl (231kB)
    100% |████████████████████████████████| 235kB 20.0MB/s ta 0:00:01
Requirement already satisfied: scipy==0.19.1 in /opt/conda/lib/python3.6/
site-packages (from -r requirements.txt (line 5)) (0.19.1)
Collecting statsmodels==0.9.0 (from -r requirements.txt (line 6))
  Downloading https://files.pythonhosted.org/packages/85/d1/69ee7e757f657
e7f527cbf500ec2d295396e5bcec873cf4eb68962c41024/statsmodels-0.9.0-cp36-cp
36m-manylinux1_x86_64.whl (7.4MB)
    100% |████████████████████████████████| 7.4MB 6.4MB/s eta 0:00:01
75% |████████████████████████        | 5.6MB 29.8MB/s eta 0:00:01
Collecting tqdm==4.19.5 (from -r requirements.txt (line 7))
  Downloading https://files.pythonhosted.org/packages/71/3c/341b4fa23cb3a
bc335207dba057c790f3bb329f6757e1fcd5d347bcf8308/tqdm-4.19.5-py2.py3-none-
any.whl (51kB)
    100% |████████████████████████████████| 61kB 12.2MB/s ta 0:00:01
Requirement already satisfied: six>=1.10 in /opt/conda/lib/python3.6/site
-packages (from matplotlib==2.1.0->-r requirements.txt (line 1)) (1.11.0)
Requirement already satisfied: python-dateutil>=2.0 in /opt/conda/lib/pyt
hon3.6/site-packages (from matplotlib==2.1.0->-r requirements.txt (line
1)) (2.6.1)
Requirement already satisfied: pytz in /opt/conda/lib/python3.6/site-pack
ages (from matplotlib==2.1.0->-r requirements.txt (line 1)) (2017.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.6/s
ite-packages/cycler-0.10.0-py3.6.egg (from matplotlib==2.1.0->-r requirem
ents.txt (line 1)) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 i
n /opt/conda/lib/python3.6/site-packages (from matplotlib==2.1.0->-r requ
irements.txt (line 1)) (2.2.0)
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, pandas, patsy, statsmodels, tqdm
  Found existing installation: numpy 1.12.1
    Uninstalling numpy-1.12.1:
      Successfully uninstalled numpy-1.12.1
  Found existing installation: pandas 0.23.3
    Uninstalling pandas-0.23.3:
      Successfully uninstalled pandas-0.23.3
  Found existing installation: patsy 0.4.1
    Uninstalling patsy-0.4.1:
      Successfully uninstalled patsy-0.4.1
  Found existing installation: statsmodels 0.8.0
    Uninstalling statsmodels-0.8.0:
      Successfully uninstalled statsmodels-0.8.0
  Found existing installation: tqdm 4.11.2
    Uninstalling tqdm-4.11.2:
      Successfully uninstalled tqdm-4.11.2
Successfully installed numpy-1.16.1 pandas-0.24.1 patsy-0.5.1 statsmodels
-0.9.0 tqdm-4.19.5
```

Load Packages

In [2]:
```
import scipy
import patsy
import pickle

import numpy as np
import pandas as pd

import scipy.sparse
import matplotlib.pyplot as plt

from statistics import median
from scipy.stats import gaussian_kde
from statsmodels.formula.api import ols
from tqdm import tqdm
```

Load Data

We'll be using the Barra dataset to get factors that can be used to predict risk. Loading and parsing the raw Barra data can be a very slow process that can significantly slow down your backtesting. For this reason, it's important to pre-process the data beforehand. For your convenience, the Barra data has already been pre-processed for you and saved into pickle files. You will load the Barra data from these pickle files.

In the code below, we start by loading `2004` factor data from the `pandas-frames.2004.pickle` file. We also load the `2003` and `2004` covariance data from the `covaraince.2003.pickle` and `covaraince.2004.pickle` files. You are encouraged to customize the data range for your backtest. For example, we recommend starting with two or three years of factor data. Remember that the covariance data should include all the years that you choose for the factor data, and also one year earlier. For example, in the code below we are using `2004` factor data, therefore, we must include `2004` in our covariance data, but also the previous year, `2003`. If you don't remember why must include this previous year, feel free to review the lessons.

In [3]:
```
barra_dir = '../../data/project_8_barra/'

data = {}
for year in [2004]:
    fil = barra_dir + "pandas-frames." + str(year) + ".pickle"
    data.update(pickle.load( open( fil, "rb" ) ))

covariance = {}
for year in [2004]:
    fil = barra_dir + "covariance." + str(year) + ".pickle"
    covariance.update(pickle.load( open(fil, "rb" ) ))

daily_return = {}
for year in [2004, 2005]:
    fil = barra_dir + "price." + str(year) + ".pickle"
    daily_return.update(pickle.load( open(fil, "rb" ) ))
```

Shift Daily Returns Data (TODO)

In the cell below, we want to incorporate a realistic time delay that exists in live trading, we'll use a two day delay for the `daily_return` data. That means the `daily_return` should be two days after the data in `data` and `cov_data`. Combine `daily_return` and `data` together in a dict called `frames`.

Since reporting of PnL is usually for the date of the returns, make sure to use the two day delay dates (dates that match the `daily_return`) when building `frames`. This means calling `frames['20040108']` will get you the prices from "20040108" and the data from `data` at "20040106".

Note: We're not shifting `covariance`, since we'll use the "DataDate" field in `frames` to lookup the covariance data. The "DataDate" field contains the date when the `data` in `frames` was recorded. For example, `frames['20040108']` will give you a value of "20040106" for the field "DataDate".

In [4]:
```
frames ={}
dlyreturn_n_days_delay = 2

# TODO: Implement
# The Student Hub helped me to figure out what was required here:
# https://hub.udacity.com/rooms/community:nd880:346730-project-581?contex
# Post by Bryant M. on March 26th, 2019, and replies to his post

date_shifts = zip(
        sorted(data.keys()),
        sorted(daily_return.keys())[dlyreturn_n_days_delay:len(data) + dl

for data_date, price_date in date_shifts:
    frames[price_date] = data[data_date].merge(daily_return[price_date],
```

In [6]:
```
print(frames['20040202'])
```

```
         Barrid  USFASTD_1DREVRSL  USFASTD_AERODEF  USFASTD_AIRLINES  \
0       USA0001            -0.453            0.000               0.0
1       USA0011             0.298            0.000               0.0
2       USA0031             0.562            0.000               0.0
3       USA0062            -0.339            0.431               0.0
4       USA00E2            -0.069            0.000               0.0
5       USA00F1             0.576            0.000               0.0
6       USA00G2            -0.399            0.000               0.0
7       USA00H1             1.029            0.000               0.0
8       USA00I1            -0.869            0.000               0.0
9       USA00J1             0.710            0.000               0.0
10      USA00K1            -0.607            0.000               0.0
11      USA00P1            -0.285            0.000               0.0
12      USA00R1             0.346            0.000               0.0
13      USA00S1            -0.545            0.000               0.0
14      USA00V1             1.175            0.000               0.0
15      USA0131            -0.085            1.000               0.0
16      USA0161            -2.587            1.000               0.0
17      USA01I1             0.776            0.000               0.0
18      USA01J2            -0.999            0.000               0.0
19      USA01L1            -0.336            0.000               0.0
20      USA01P1            -0.879            0.000               0.0
21      USA01Q1             0.623            0.000               0.0
22      USA0202            -0.638            0.000               0.0
23      USA0231            -0.284            0.000               0.0
24      USA0281            -0.017            0.000               0.0
25      USA0291            -0.720            0.000               0.0
26      USA02A1             0.191            0.000               0.0
27      USA02B1            -0.915            0.000               0.0
28      USA02H1             0.136            0.000               0.0
29      USA02P1            -0.295            0.000               0.0
...         ...               ...              ...               ...
12387   USAZY41             0.246            0.000               0.0
12388   USAZY51            -1.058            0.000               0.0
12389   USAZY61            -0.230            0.000               0.0
12390   USAZY71            -2.661            0.000               0.0
12391   USAZYI1             1.538            0.000               0.0
12392   USAZYJ1            -1.280            0.000               0.0
12393   USAZYK1            -0.852            0.000               0.0
12394   USAZYM1            -0.771            0.000               0.0
12395   USAZYR1            -2.661            0.000               0.0
12396   USAZYS1            -1.242            0.000               0.0
12397   USAZYT1            -0.049            0.000               0.0
12398   USAZYV1             2.089            0.000               0.0
12399   USAZYX1            -0.625            0.000               0.0
12400   USAZZ41            -0.788            0.000               0.0
12401   USAZZ51            -1.226            0.000               0.0
12402   USAZZ61            -1.172            0.000               0.0
12403   USAZZ71            -1.659            0.000               0.0
12404   USAZZ81             0.136            0.000               0.0
12405   USAZZ91            -1.684            0.000               0.0
12406   USAZZA1             2.349            0.000               0.0
12407   USAZZB1            -1.159            0.000               0.0
12408   USAZZD1            -0.671            0.000               0.0
```

12409	USAZZF1	-2.661	0.000	0.0
12410	USAZZI1	2.664	0.000	0.0
12411	USAZZJ1	-2.661	0.000	0.0
12412	USAZZL1	-0.470	0.000	0.0
12413	USAZZP1	-0.117	0.000	0.0
12414	USAZZR1	-0.990	0.000	0.0
12415	USAZZX1	-1.602	0.000	0.0
12416	USAZZY1	-0.299	0.000	0.0

	USFASTD_ALUMSTEL	USFASTD_APPAREL	USFASTD_AUTO	USFASTD_BANKS \
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0
16	0.0	0.0	0.0	0.0
17	0.0	0.0	0.0	0.0
18	0.0	0.0	0.0	0.0
19	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0
21	0.0	0.0	0.0	0.0
22	0.0	0.0	0.0	0.0
23	0.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0
25	0.0	0.0	0.0	0.0
26	0.0	0.0	0.0	0.0
27	0.0	0.0	0.0	0.0
28	0.0	0.0	0.0	1.0
29	0.0	0.0	0.0	0.0
...	...	...	...	...
12387	0.0	0.0	0.0	0.0
12388	0.0	0.0	0.0	0.0
12389	0.0	0.0	0.0	0.0
12390	0.0	0.0	0.0	0.0
12391	0.0	0.0	0.0	0.0
12392	0.0	0.0	0.0	0.0
12393	0.0	0.0	0.0	0.0
12394	0.0	0.0	0.0	0.0
12395	0.0	0.0	0.0	0.0
12396	0.0	0.0	0.0	0.0
12397	0.0	0.0	0.0	0.0
12398	0.0	0.0	0.0	0.0
12399	0.0	0.0	0.0	0.0
12400	0.0	0.0	0.0	0.0
12401	0.0	0.0	0.0	0.0
12402	0.0	0.0	0.0	0.0

```
12403               0.0              0.0              0.0              0.0
12404               0.0              0.0              0.0              0.0
12405               0.0              0.0              0.0              0.0
12406               0.0              0.0              0.0              0.0
12407               0.0              0.0              0.0              0.0
12408               0.0              0.0              0.0              0.0
12409               0.0              0.0              0.0              0.0
12410               0.0              0.0              0.0              0.0
12411               0.0              0.0              0.0              0.0
12412               0.0              0.0              0.0              0.0
12413               0.0              0.0              0.0              0.0
12414               0.0              0.0              0.0              0.0
12415               0.0              0.0              0.0              0.0
12416               0.0              0.0              0.0              0.0

       USFASTD_BETA  USFASTD_BEVTOB  ...  DailyVolume   ADTCA_30  \
0            -2.132           0.000  ...          NaN        NaN
1            -2.132           0.000  ...          NaN        NaN
2            -2.034           0.000  ...          NaN        NaN
3            -2.268           0.000  ...          NaN        NaN
4            -2.159           0.000  ...          NaN        NaN
5            -2.042           0.000  ...          NaN        NaN
6            -2.137           0.000  ...          NaN        NaN
7            -1.875           0.000  ...          NaN        NaN
8            -0.714           0.000  ...          NaN        NaN
9            -2.017           0.863  ...          NaN        NaN
10           -2.280           0.000  ...          NaN        NaN
11           -1.060           0.000  ...          NaN        NaN
12           -1.683           0.000  ...          NaN        NaN
13           -1.104           0.000  ...      17526.0        NaN
14           -2.132           0.000  ...          NaN        NaN
15           -2.410           0.000  ...          NaN        NaN
16           -1.102           0.000  ...       1600.0        NaN
17           -1.929           0.000  ...          NaN        NaN
18           -2.003           0.000  ...          NaN        NaN
19           -1.987           0.000  ...          NaN        NaN
20           -1.743           0.000  ...          NaN        NaN
21           -2.564           1.000  ...          NaN        NaN
22           -0.328           0.000  ...          NaN        NaN
23           -1.961           0.000  ...          NaN        NaN
24           -2.144           0.000  ...          NaN        NaN
25           -1.252           0.000  ...          NaN        NaN
26           -2.101           0.000  ...          NaN        NaN
27           -1.391           0.000  ...        205.0        NaN
28           -1.479           0.000  ...          NaN        NaN
29           -2.131           0.000  ...      18250.0        NaN
...             ...             ...  ...          ...        ...
12387        -2.132           0.000  ...          NaN        NaN
12388        -1.429           0.000  ...       7500.0    1204.44
12389        -0.721           0.000  ...          NaN        NaN
12390         0.319           0.000  ...      18000.0     196.03
12391        -0.727           0.000  ...      14000.0    6070.63
12392        -0.741           0.000  ...          NaN        NaN
12393         0.020           0.000  ...       1000.0      38.26
12394        -0.121           0.000  ...          NaN        NaN
12395        -0.656           0.000  ...    1088748.0  302807.87
12396         0.165           0.000  ...          NaN        NaN
```

```
12397         -1.217            0.000  ...     133730.0    47196.41
12398         -0.647            0.000  ...    2236000.0    12660.04
12399         -1.294            0.000  ...        500.0    17031.25
12400         -1.307            0.000  ...     100400.0    18613.46
12401         -1.327            0.000  ...          NaN         NaN
12402         -0.646            0.000  ...     349844.0     6387.69
12403         -0.783            0.000  ...          NaN         NaN
12404         -1.604            1.000  ...      19800.0  1642336.25
12405          0.057            0.000  ...          NaN         NaN
12406         -0.137            0.000  ...       6664.0     2648.89
12407         -1.649            0.000  ...          NaN         NaN
12408          0.342            0.000  ...          NaN         NaN
12409         -0.105            0.000  ...       4000.0     2294.39
12410         -1.827            1.000  ...      15000.0      954.60
12411         -1.579            0.000  ...          0.0         NaN
12412         -0.341            0.000  ...       3000.0    16052.11
12413         -1.012            1.000  ...          NaN         NaN
12414         -0.689            0.000  ...          NaN         NaN
12415         -0.033            0.000  ...      10000.0     4573.75
12416         -0.549            0.000  ...     110500.0  3897348.45

       IssuerMarketCap     Yield   TotalRisk    SpecRisk  HistBeta  PredB
eta  \
0         5.289123e+10  0.188679   20.694434   14.729043 -0.000178  0.115
036
1         5.961760e+09  0.000000   23.609017   17.014409  0.000004  0.108
720
2         6.836196e+10  2.103004   28.434662   23.925639  0.046058  0.194
914
3         3.091896e+10  2.243494   33.123394   30.448789 -0.064070  0.223
348
4         5.498100e+10  2.167256   38.742879   35.090504 -0.012908  0.283
564
5         1.807170e+11  0.392275   20.908445   16.315682  0.042024  0.306
262
6         1.543809e+10  4.679803   29.128388   20.217000 -0.002480 -0.010
724
7         8.388786e+10  0.000000   31.290932   25.358922  0.120849  0.244
778
8         1.748390e+10  0.087623   43.596154   34.359011  0.665980  1.096
755
9         4.224772e+10  1.670463   29.970976   26.739438  0.053856  0.292
584
10        3.582410e+10  0.000000   46.333840   36.759635 -0.069733  0.509
957
11        2.563947e+10  1.413784   26.625234   21.618835  0.503530  0.880
865
12        6.628188e+09  2.641914   22.013825   16.586309  0.210873  0.598
267
13        1.479395e+10  1.244629   29.702855   26.651338  0.482919  0.576
354
14        1.139173e+10  1.458435   18.855765   13.979951  0.000004  0.090
631
15        3.384689e+10  1.613669   33.266556   28.789954 -0.130689  0.145
135
16        3.384689e+10  1.613669   41.551565   37.197262  0.483906  0.696
619
```

```
17        2.857268e+10  0.196207   20.486961   16.531848  0.095167  0.183
195
18        2.665842e+10  1.782703   31.059667   27.220645  0.060486  0.421
393
19        9.331422e+10  3.380021   31.157916   26.862454  0.068120  0.215
968
20        1.807344e+11  0.000000   46.490068   41.729253  0.182851  0.384
355
21        1.218813e+11  1.733990   29.504052   25.863022 -0.203113  0.067
142
22        1.614523e+10  0.000000   40.607774   34.767773  0.847529  1.017
267
23        8.649954e+10  0.000000   32.091533   28.309044  0.080463  0.449
086
24        4.376922e+10  3.050774   26.983458   22.155723 -0.005764  0.347
621
25        3.480358e+10  0.000000   31.818067   27.290722  0.413301  0.718
060
26        8.283299e+09  1.947799   15.724159   11.585231  0.014474  0.230
623
27        8.283299e+09  1.947799   47.468973   44.602211  0.348162  0.696
023
28        7.470167e+10  2.398082   28.189902   24.433436  0.306536  0.527
372
29        3.409415e+11  2.582311   30.580899   27.093828  0.000217  0.275
863
...                ...       ...         ...         ...       ...
...
12387     5.056250e+06       NaN   27.260771   18.932814  0.000004 -0.023
863
12388     6.090515e+06  0.000000  104.469478  101.750333  0.330220  0.593
409
12389     3.570880e+06       NaN   97.702577   94.360826  0.662657  0.913
293
12390     4.904000e+05  0.000000  133.526537  129.376135  1.151356  1.328
151
12391     6.241790e+06  0.000000  110.837739  107.382513  0.659717  0.966
192
12392     2.165040e+07       NaN  101.811941   97.978433  0.653215  1.072
332
12393     6.832300e+04       NaN  103.543000   99.234095  1.011012  1.276
880
12394     1.513400e+06       NaN  112.477902  108.892673  0.944760  1.100
323
12395     1.934550e+07  0.000000  161.471931  158.735624  0.693495  0.823
788
12396     5.030700e+01  0.000000   54.510589   43.898056  1.078707  1.314
595
12397     2.664900e+07  0.000000  101.664223   98.931488  0.429639  0.725
879
12398     2.022045e+06  0.000000  163.088286  160.367777  0.697482  1.088
300
12399     4.408008e+07  0.000000   36.575626   31.719218  0.393721  0.240
775
12400     5.667890e+06       NaN   97.804564   94.667639  0.387307  0.706
172
12401     7.414000e+05       NaN   48.075393   40.713728  0.377943  0.628
```

```
380
12402     7.785400e+06  0.000000  184.516847  182.334727  0.697834  0.899
932
12403     1.550796e+07  0.000000   94.508199   89.964068  0.633557  0.928
682
12404     2.951147e+10  4.714286   32.947040   30.112542  0.247859  0.371
527
12405     1.352860e+06  0.000000   61.704020   51.280116  1.028419  1.074
434
12406     1.087100e+06       NaN  130.599161  126.879075  0.936892  1.241
884
12407     1.251480e+06  0.000000   41.244786   33.040499  0.226796  0.535
050
12408     2.591100e+05       NaN  116.403801  112.333291  1.162148  1.408
065
12409     3.492480e+06       NaN  138.440849  134.845868  0.952318  1.176
245
12410     5.862500e+05  0.000000  200.094389  198.665668  0.143345  0.409
695
12411     1.081953e+07  0.000000   96.897413   91.863688  0.259780  0.335
687
12412     3.675600e+07  0.000000   78.220357   73.194224  0.841392  1.083
367
12413     1.307280e+06       NaN  112.223384  109.112273  0.525998  0.748
585
12414     2.437080e+07       NaN   52.605311   45.356111  0.677706  0.911
724
12415     2.031860e+06  0.000000  172.922415  170.021760  0.986007  1.199
472
12416     8.473052e+08  0.620877   49.819346   37.558330  0.743676  0.979
055

       DataDate  DlyReturn
0      20040129   0.000000
1      20040129   0.000000
2      20040129   0.000000
3      20040129   0.000000
4      20040129   0.000000
5      20040129   0.000000
6      20040129   0.000000
7      20040129   0.000000
8      20040129   0.000000
9      20040129   0.000000
10     20040129   0.000000
11     20040129   0.000000
12     20040129   0.000000
13     20040129  -0.028063
14     20040129   0.000000
15     20040129   0.000000
16     20040129  -0.007407
17     20040129   0.000000
18     20040129   0.000000
19     20040129   0.000000
20     20040129   0.000000
21     20040129   0.000000
22     20040129   0.000000
23     20040129   0.000000
```

```
24     20040129   0.000000
25     20040129   0.000000
26     20040129   0.000000
27     20040129  -0.044211
28     20040129   0.000000
29     20040129   0.012723
...         ...        ...
12387  20040129   0.000000
12388  20040129   0.090047
12389  20040129   0.000000
12390  20040129  -0.500000
12391  20040129   0.126582
12392  20040129   0.000000
12393  20040129   0.000000
12394  20040129   0.000000
12395  20040129  -0.037736
12396  20040129   0.000000
12397  20040129  -0.033898
12398  20040129   0.000000
12399  20040129  -0.004711
12400  20040129   0.000000
12401  20040129   0.000000
12402  20040129   0.000000
12403  20040129   0.346154
12404  20040129   0.010646
12405  20040129   0.000000
12406  20040129   0.800000
12407  20040129   0.000000
12408  20040129   0.000000
12409  20040129   0.000000
12410  20040129   0.000000
12411  20040129  -0.027843
12412  20040129  -0.066390
12413  20040129   0.000000
12414  20040129   0.000000
12415  20040129   0.000000
12416  20040129  -0.003458

[12417 rows x 93 columns]
```

Add Daily Returns date column (Optional)

Name the column `DlyReturnDate`. **Hint**: create a list containing copies of the date, then create a pandas series.

In [7]:
```
# Optional
for DlyReturnDate, df in frames.items():
    # Get the number of rows
    n_rows = df.shape[0]
    # Add a column for the datadate
    df['DlyReturnDate'] = pd.Series([DlyReturnDate] * n_rows)
```

In [8]:
```
print(frames['20040202'])
```

	Barrid	USFASTD_1DREVRSL	USFASTD_AERODEF	USFASTD_AIRLINES	\
0	USA0001	-0.453	0.000	0.0	
1	USA0011	0.298	0.000	0.0	
2	USA0031	0.562	0.000	0.0	
3	USA0062	-0.339	0.431	0.0	
4	USA00E2	-0.069	0.000	0.0	
5	USA00F1	0.576	0.000	0.0	
6	USA00G2	-0.399	0.000	0.0	
7	USA00H1	1.029	0.000	0.0	
8	USA00I1	-0.869	0.000	0.0	
9	USA00J1	0.710	0.000	0.0	
10	USA00K1	-0.607	0.000	0.0	
11	USA00P1	-0.285	0.000	0.0	
12	USA00R1	0.346	0.000	0.0	
13	USA00S1	-0.545	0.000	0.0	
14	USA00V1	1.175	0.000	0.0	
15	USA0131	-0.085	1.000	0.0	
16	USA0161	-2.587	1.000	0.0	
17	USA01I1	0.776	0.000	0.0	
18	USA01J2	-0.999	0.000	0.0	
19	USA01L1	-0.336	0.000	0.0	
20	USA01P1	-0.879	0.000	0.0	
21	USA01Q1	0.623	0.000	0.0	
22	USA0202	-0.638	0.000	0.0	
23	USA0231	-0.284	0.000	0.0	
24	USA0281	-0.017	0.000	0.0	
25	USA0291	-0.720	0.000	0.0	
26	USA02A1	0.191	0.000	0.0	
27	USA02B1	-0.915	0.000	0.0	
28	USA02H1	0.136	0.000	0.0	
29	USA02P1	-0.295	0.000	0.0	
...	...	...	...	...	
12387	USAZY41	0.246	0.000	0.0	
12388	USAZY51	-1.058	0.000	0.0	
12389	USAZY61	-0.230	0.000	0.0	
12390	USAZY71	-2.661	0.000	0.0	
12391	USAZYI1	1.538	0.000	0.0	
12392	USAZYJ1	-1.280	0.000	0.0	
12393	USAZYK1	-0.852	0.000	0.0	
12394	USAZYM1	-0.771	0.000	0.0	
12395	USAZYR1	-2.661	0.000	0.0	
12396	USAZYS1	-1.242	0.000	0.0	
12397	USAZYT1	-0.049	0.000	0.0	
12398	USAZYV1	2.089	0.000	0.0	
12399	USAZYX1	-0.625	0.000	0.0	
12400	USAZZ41	-0.788	0.000	0.0	
12401	USAZZ51	-1.226	0.000	0.0	
12402	USAZZ61	-1.172	0.000	0.0	
12403	USAZZ71	-1.659	0.000	0.0	
12404	USAZZ81	0.136	0.000	0.0	
12405	USAZZ91	-1.684	0.000	0.0	
12406	USAZZA1	2.349	0.000	0.0	
12407	USAZZB1	-1.159	0.000	0.0	
12408	USAZZD1	-0.671	0.000	0.0	
12409	USAZZF1	-2.661	0.000	0.0	
12410	USAZZI1	2.664	0.000	0.0	
12411	USAZZJ1	-2.661	0.000	0.0	

12412	USAZZL1	-0.470	0.000	0.0
12413	USAZZP1	-0.117	0.000	0.0
12414	USAZZR1	-0.990	0.000	0.0
12415	USAZZX1	-1.602	0.000	0.0
12416	USAZZY1	-0.299	0.000	0.0

	USFASTD_ALUMSTEL	USFASTD_APPAREL	USFASTD_AUTO	USFASTD_BANKS \
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0
16	0.0	0.0	0.0	0.0
17	0.0	0.0	0.0	0.0
18	0.0	0.0	0.0	0.0
19	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0
21	0.0	0.0	0.0	0.0
22	0.0	0.0	0.0	0.0
23	0.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0
25	0.0	0.0	0.0	0.0
26	0.0	0.0	0.0	0.0
27	0.0	0.0	0.0	0.0
28	0.0	0.0	0.0	1.0
29	0.0	0.0	0.0	0.0
...	...	...	...	...
12387	0.0	0.0	0.0	0.0
12388	0.0	0.0	0.0	0.0
12389	0.0	0.0	0.0	0.0
12390	0.0	0.0	0.0	0.0
12391	0.0	0.0	0.0	0.0
12392	0.0	0.0	0.0	0.0
12393	0.0	0.0	0.0	0.0
12394	0.0	0.0	0.0	0.0
12395	0.0	0.0	0.0	0.0
12396	0.0	0.0	0.0	0.0
12397	0.0	0.0	0.0	0.0
12398	0.0	0.0	0.0	0.0
12399	0.0	0.0	0.0	0.0
12400	0.0	0.0	0.0	0.0
12401	0.0	0.0	0.0	0.0
12402	0.0	0.0	0.0	0.0
12403	0.0	0.0	0.0	0.0
12404	0.0	0.0	0.0	0.0
12405	0.0	0.0	0.0	0.0

```
12406             0.0              0.0              0.0              0.0
12407             0.0              0.0              0.0              0.0
12408             0.0              0.0              0.0              0.0
12409             0.0              0.0              0.0              0.0
12410             0.0              0.0              0.0              0.0
12411             0.0              0.0              0.0              0.0
12412             0.0              0.0              0.0              0.0
12413             0.0              0.0              0.0              0.0
12414             0.0              0.0              0.0              0.0
12415             0.0              0.0              0.0              0.0
12416             0.0              0.0              0.0              0.0

       USFASTD_BETA  USFASTD_BEVTOB  ...   ADTCA_30  IssuerMarketCap  \
0            -2.132           0.000  ...        NaN     5.289123e+10
1            -2.132           0.000  ...        NaN     5.961760e+09
2            -2.034           0.000  ...        NaN     6.836196e+10
3            -2.268           0.000  ...        NaN     3.091896e+10
4            -2.159           0.000  ...        NaN     5.498100e+10
5            -2.042           0.000  ...        NaN     1.807170e+11
6            -2.137           0.000  ...        NaN     1.543809e+10
7            -1.875           0.000  ...        NaN     8.388786e+10
8            -0.714           0.000  ...        NaN     1.748390e+10
9            -2.017           0.863  ...        NaN     4.224772e+10
10           -2.280           0.000  ...        NaN     3.582410e+10
11           -1.060           0.000  ...        NaN     2.563947e+10
12           -1.683           0.000  ...        NaN     6.628188e+09
13           -1.104           0.000  ...        NaN     1.479395e+10
14           -2.132           0.000  ...        NaN     1.139173e+10
15           -2.410           0.000  ...        NaN     3.384689e+10
16           -1.102           0.000  ...        NaN     3.384689e+10
17           -1.929           0.000  ...        NaN     2.857268e+10
18           -2.003           0.000  ...        NaN     2.665842e+10
19           -1.987           0.000  ...        NaN     9.331422e+10
20           -1.743           0.000  ...        NaN     1.807344e+11
21           -2.564           1.000  ...        NaN     1.218813e+11
22           -0.328           0.000  ...        NaN     1.614523e+10
23           -1.961           0.000  ...        NaN     8.649954e+10
24           -2.144           0.000  ...        NaN     4.376922e+10
25           -1.252           0.000  ...        NaN     3.480358e+10
26           -2.101           0.000  ...        NaN     8.283299e+09
27           -1.391           0.000  ...        NaN     8.283299e+09
28           -1.479           0.000  ...        NaN     7.470167e+10
29           -2.131           0.000  ...        NaN     3.409415e+11
...             ...             ...  ...        ...              ...
12387        -2.132           0.000  ...        NaN     5.056250e+06
12388        -1.429           0.000  ...    1204.44     6.090515e+06
12389        -0.721           0.000  ...        NaN     3.570880e+06
12390         0.319           0.000  ...     196.03     4.904000e+05
12391        -0.727           0.000  ...    6070.63     6.241790e+06
12392        -0.741           0.000  ...        NaN     2.165040e+07
12393         0.020           0.000  ...      38.26     6.832300e+04
12394        -0.121           0.000  ...        NaN     1.513400e+06
12395        -0.656           0.000  ...  302807.87     1.934550e+07
12396         0.165           0.000  ...        NaN     5.030700e+01
12397        -1.217           0.000  ...   47196.41     2.664900e+07
12398        -0.647           0.000  ...   12660.04     2.022045e+06
12399        -1.294           0.000  ...   17031.25     4.408008e+07
```

```
12400        -1.307            0.000  ...    18613.46     5.667890e+06
12401        -1.327            0.000  ...         NaN     7.414000e+05
12402        -0.646            0.000  ...     6387.69     7.785400e+06
12403        -0.783            0.000  ...         NaN     1.550796e+07
12404        -1.604            1.000  ...  1642336.25     2.951147e+10
12405         0.057            0.000  ...         NaN     1.352860e+06
12406        -0.137            0.000  ...     2648.89     1.087100e+06
12407        -1.649            0.000  ...         NaN     1.251480e+06
12408         0.342            0.000  ...         NaN     2.591100e+05
12409        -0.105            0.000  ...     2294.39     3.492480e+06
12410        -1.827            1.000  ...      954.60     5.862500e+05
12411        -1.579            0.000  ...         NaN     1.081953e+07
12412        -0.341            0.000  ...    16052.11     3.675600e+07
12413        -1.012            1.000  ...         NaN     1.307280e+06
12414        -0.689            0.000  ...         NaN     2.437080e+07
12415        -0.033            0.000  ...     4573.75     2.031860e+06
12416        -0.549            0.000  ...  3897348.45     8.473052e+08

          Yield  TotalRisk    SpecRisk  HistBeta  PredBeta  DataDate  \
0      0.188679  20.694434   14.729043 -0.000178  0.115036  20040129
1      0.000000  23.609017   17.014409  0.000004  0.108720  20040129
2      2.103004  28.434662   23.925639  0.046058  0.194914  20040129
3      2.243494  33.123394   30.448789 -0.064070  0.223348  20040129
4      2.167256  38.742879   35.090504 -0.012908  0.283564  20040129
5      0.392275  20.908445   16.315682  0.042024  0.306262  20040129
6      4.679803  29.128388   20.217000 -0.002480 -0.010724  20040129
7      0.000000  31.290932   25.358922  0.120849  0.244778  20040129
8      0.087623  43.596154   34.359011  0.665980  1.096755  20040129
9      1.670463  29.970976   26.739438  0.053856  0.292584  20040129
10     0.000000  46.333840   36.759635 -0.069733  0.509957  20040129
11     1.413784  26.625234   21.618835  0.503530  0.880865  20040129
12     2.641914  22.013825   16.586309  0.210873  0.598267  20040129
13     1.244629  29.702855   26.651338  0.482919  0.576354  20040129
14     1.458435  18.855765   13.979951  0.000004  0.090631  20040129
15     1.613669  33.266556   28.789954 -0.130689  0.145135  20040129
16     1.613669  41.551565   37.197262  0.483906  0.696619  20040129
17     0.196207  20.486961   16.531848  0.095167  0.183195  20040129
18     1.782703  31.059667   27.220645  0.060486  0.421393  20040129
19     3.380021  31.157916   26.862454  0.068120  0.215968  20040129
20     0.000000  46.490068   41.729253  0.182851  0.384355  20040129
21     1.733990  29.504052   25.863022 -0.203113  0.067142  20040129
22     0.000000  40.607774   34.767773  0.847529  1.017267  20040129
23     0.000000  32.091533   28.309044  0.080463  0.449086  20040129
24     3.050774  26.983458   22.155723 -0.005764  0.347621  20040129
25     0.000000  31.818067   27.290722  0.413301  0.718060  20040129
26     1.947799  15.724159   11.585231  0.014474  0.230623  20040129
27     1.947799  47.468973   44.602211  0.348162  0.696023  20040129
28     2.398082  28.189902   24.433436  0.306536  0.527372  20040129
29     2.582311  30.580899   27.093828  0.000217  0.275863  20040129
...         ...        ...         ...       ...       ...       ...
12387       NaN  27.260771   18.932814  0.000004 -0.023863  20040129
12388  0.000000 104.469478  101.750333  0.330220  0.593409  20040129
12389       NaN  97.702577   94.360826  0.662657  0.913293  20040129
12390  0.000000 133.526537  129.376135  1.151356  1.328151  20040129
12391  0.000000 110.837739  107.382513  0.659717  0.966192  20040129
12392       NaN 101.811941   97.978433  0.653215  1.072332  20040129
12393       NaN 103.543000   99.234095  1.011012  1.276880  20040129
```

```
12394       NaN  112.477902  108.892673  0.944760  1.100323  20040129
12395  0.000000  161.471931  158.735624  0.693495  0.823788  20040129
12396  0.000000   54.510589   43.898056  1.078707  1.314595  20040129
12397  0.000000  101.664223   98.931488  0.429639  0.725879  20040129
12398  0.000000  163.088286  160.367777  0.697482  1.088300  20040129
12399  0.000000   36.575626   31.719218  0.393721  0.240775  20040129
12400       NaN   97.804564   94.667639  0.387307  0.706172  20040129
12401       NaN   48.075393   40.713728  0.377943  0.628380  20040129
12402  0.000000  184.516847  182.334727  0.697834  0.899932  20040129
12403  0.000000   94.508199   89.964068  0.633557  0.928682  20040129
12404  4.714286   32.947040   30.112542  0.247859  0.371527  20040129
12405  0.000000   61.704020   51.280116  1.028419  1.074434  20040129
12406       NaN  130.599161  126.879075  0.936892  1.241884  20040129
12407  0.000000   41.244786   33.040499  0.226796  0.535050  20040129
12408       NaN  116.403801  112.333291  1.162148  1.408065  20040129
12409       NaN  138.440849  134.845868  0.952318  1.176245  20040129
12410  0.000000  200.094389  198.665668  0.143345  0.409695  20040129
12411  0.000000   96.897413   91.863688  0.259780  0.335687  20040129
12412  0.000000   78.220357   73.194224  0.841392  1.083367  20040129
12413       NaN  112.223384  109.112273  0.525998  0.748585  20040129
12414       NaN   52.605311   45.356111  0.677706  0.911724  20040129
12415  0.000000  172.922415  170.021760  0.986007  1.199472  20040129
12416  0.620877   49.819346   37.558330  0.743676  0.979055  20040129

       DlyReturn  DlyReturnDate
0       0.000000       20040202
1       0.000000       20040202
2       0.000000       20040202
3       0.000000       20040202
4       0.000000       20040202
5       0.000000       20040202
6       0.000000       20040202
7       0.000000       20040202
8       0.000000       20040202
9       0.000000       20040202
10      0.000000       20040202
11      0.000000       20040202
12      0.000000       20040202
13     -0.028063       20040202
14      0.000000       20040202
15      0.000000       20040202
16     -0.007407       20040202
17      0.000000       20040202
18      0.000000       20040202
19      0.000000       20040202
20      0.000000       20040202
21      0.000000       20040202
22      0.000000       20040202
23      0.000000       20040202
24      0.000000       20040202
25      0.000000       20040202
26      0.000000       20040202
27     -0.044211       20040202
28      0.000000       20040202
29      0.012723       20040202
...          ...            ...
12387   0.000000       20040202
```

```
12388   0.090047        20040202
12389   0.000000        20040202
12390  -0.500000        20040202
12391   0.126582        20040202
12392   0.000000        20040202
12393   0.000000        20040202
12394   0.000000        20040202
12395  -0.037736        20040202
12396   0.000000        20040202
12397  -0.033898        20040202
12398   0.000000        20040202
12399  -0.004711        20040202
12400   0.000000        20040202
12401   0.000000        20040202
12402   0.000000        20040202
12403   0.346154        20040202
12404   0.010646        20040202
12405   0.000000        20040202
12406   0.800000        20040202
12407   0.000000        20040202
12408   0.000000        20040202
12409   0.000000        20040202
12410   0.000000        20040202
12411  -0.027843        20040202
12412  -0.066390        20040202
12413   0.000000        20040202
12414   0.000000        20040202
12415   0.000000        20040202
12416  -0.003458        20040202

[12417 rows x 94 columns]
```

Winsorize

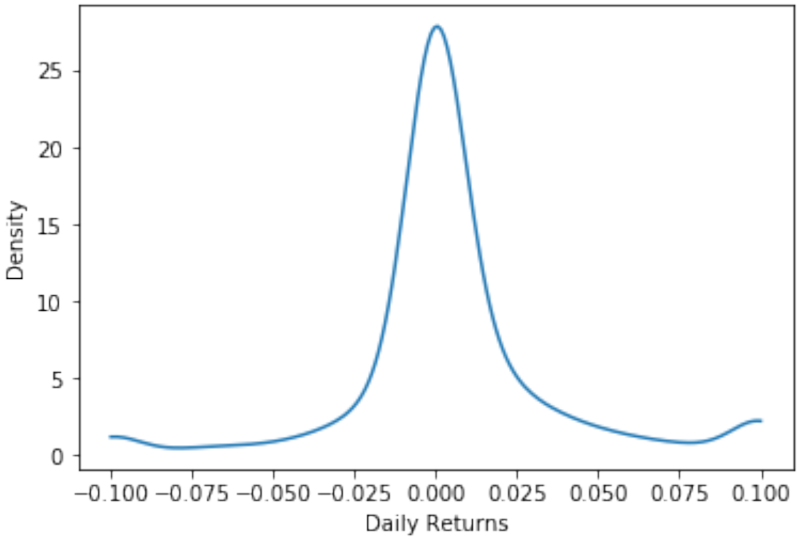
As we have done in other projects, we'll want to avoid extremely positive or negative values in our data. Will therefore create a function, `wins`, that will clip our values to a minimum and maximum range. This process is called **Winsorizing**. Remember that this helps us handle noise, which may otherwise cause unusually large positions.

In [9]:
```
def wins(x,a,b):
    return np.where(x <= a,a, np.where(x >= b, b, x))
```

Density Plot

Let's check our `wins` function by taking a look at the distribution of returns for a single day `20040102`. We will clip our data from `-0.1` to `0.1` and plot it using our `density_plot` function.

In [10]:
```
def density_plot(data):
    density = gaussian_kde(data)
    xs = np.linspace(np.min(data),np.max(data),200)
    density.covariance_factor = lambda : .25
    density._compute_covariance()
    plt.plot(xs,density(xs))
    plt.xlabel('Daily Returns')
    plt.ylabel('Density')
    plt.show()

test = frames['20040108']
test['DlyReturn'] = wins(test['DlyReturn'],-0.1,0.1)
density_plot(test['DlyReturn'])
```



Factor Exposures and Factor Returns

Recall that:

$r_{i,t} = \sum_{j=1}^{k} (\beta_{i,j,t-2} \times f_{j,t})$
where $i=1...N$ (N assets),
and $j=1...k$ (k factors).

where $r_{i,t}$ is the return, $\beta_{i,j,t-2}$ is the factor exposure, and $f_{j,t}$ is the factor return. Since we get the factor exposures from the Barra data, and we know the returns, it is possible to estimate the factor returns. In this notebook, we will use the Ordinary Least Squares (OLS) method to estimate the factor exposures, $f_{j,t}$, by using $\beta_{i,j,t-2}$ as the independent variable, and $r_{i,t}$ as the dependent variable.

In [11]:
```
def get_formula(factors, Y):
    L = ["0"]
    L.extend(factors)
    return Y + " ~ " + " + ".join(L)

def factors_from_names(n):
    return list(filter(lambda x: "USFASTD_" in x, n))

def estimate_factor_returns(df):
    ## build universe based on filters
    estu = df.loc[df.IssuerMarketCap > 1e9].copy(deep=True)

    ## winsorize returns for fitting
    estu['DlyReturn'] = wins(estu['DlyReturn'], -0.25, 0.25)

    all_factors = factors_from_names(list(df))
    form = get_formula(all_factors, "DlyReturn")
    model = ols(form, data=estu)
    results = model.fit()
    return results
```

In [12]:
```
facret = {}

for date in frames:
    facret[date] = estimate_factor_returns(frames[date]).params
```

In [13]:
```
my_dates = sorted(list(map(lambda date: pd.to_datetime(date, format='%Y%m
```

Choose Alpha Factors

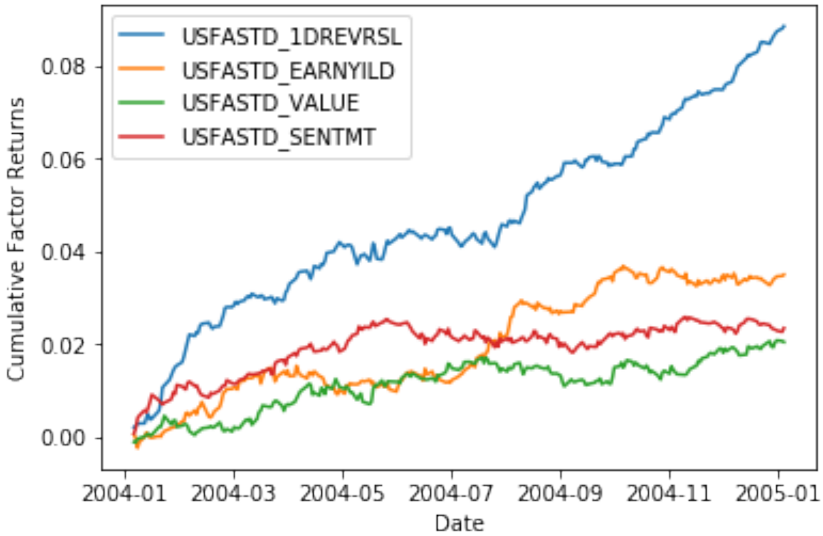
We will now choose our alpha factors. Barra's factors include some alpha factors that we have seen before, such as:

- **USFASTD_1DREVRSL** : Reversal
- **USFASTD_EARNYILD** : Earnings Yield
- **USFASTD_VALUE** : Value
- **USFASTD_SENTMT** : Sentiment

We will choose these alpha factors for now, but you are encouraged to come back to this later and try other factors as well.

In [14]:
```
alpha_factors = ["USFASTD_1DREVRSL", "USFASTD_EARNYILD", "USFASTD_VALUE",

facret_df = pd.DataFrame(index = my_dates)

for dt in my_dates:
    for alp in alpha_factors:
        facret_df.at[dt, alp] = facret[dt.strftime('%Y%m%d')][alp]

for column in facret_df.columns:
        plt.plot(facret_df[column].cumsum(), label=column)
plt.legend(loc='upper left')
plt.xlabel('Date')
plt.ylabel('Cumulative Factor Returns')
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/pandas/plotting/_converter.py:129:
FutureWarning: Using an implicitly registered datetime converter for a ma
tplotlib plotting method. The converter was registered by pandas on impor
t. Future versions of pandas will require you to explicitly register matp
lotlib converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```



Merge Previous Portfolio Holdings

In order to optimize our portfolio we will use the previous day's holdings to estimate the trade size and transaction costs. In order to keep track of the holdings from the previous day we will include a column to hold the portfolio holdings of the previous day. These holdings of all our assets will be initialized to zero when the backtest first starts.

In [15]:
```
def clean_nas(df):
    numeric_columns = df.select_dtypes(include=[np.number]).columns.tolis

    for numeric_column in numeric_columns:
        df[numeric_column] = np.nan_to_num(df[numeric_column])

    return df
```

In [16]:
```
previous_holdings = pd.DataFrame(data = {"Barrid" : ["USA02P1"], "h.opt.p
df = frames[my_dates[0].strftime('%Y%m%d')]

df = df.merge(previous_holdings, how = 'left', on = 'Barrid')
df = clean_nas(df)
df.loc[df['SpecRisk'] == 0]['SpecRisk'] = median(df['SpecRisk'])
```

Build Universe Based on Filters (TODO)

In the cell below, implement the function `get_universe` that creates a stock universe by selecting only those companies that have a market capitalization of at least 1 billion dollars **OR** that are in the previous day's holdings, even if on the current day, the company no longer meets the 1 billion dollar criteria.

When creating the universe, make sure you use the `.copy()` attribute to create a copy of the data. Also, it is very important to make sure that we are not looking at returns when forming the portfolio! to make this impossible, make sure to drop the column containing the daily return.

In [17]:
```
def get_universe(df):
    """
    Create a stock universe based on filters

    Parameters
    ----------
    df : DataFrame
        All stocks

    Returns
    -------
    universe : DataFrame
        Selected stocks based on filters
    """

    # TODO: Implement
    # Get universe
    universe = df.loc[(df['IssuerMarketCap'] >= 1e9) | (abs(df['h.opt.pre

    # Remove daily returns from df so that it is impossible to read off d
    universe = universe.drop(columns = 'DlyReturn')

    return universe

universe = get_universe(df)
```

In [18]:
```
date = str(int(universe['DataDate'][1]))
```

Factors

We will now extract both the risk factors and alpha factors. We begin by first getting all the factors using the `factors_from_names` function defined previously.

In [19]:
```
all_factors = factors_from_names(list(universe))
```

We will now create the function `setdiff` to just select the factors that we have not defined as alpha factors

In [20]:
```
def setdiff(temp1, temp2):
    s = set(temp2)
    temp3 = [x for x in temp1 if x not in s]
    return temp3
```

In [21]:
```
risk_factors = setdiff(all_factors, alpha_factors)
```

We will also save the column that contains the previous holdings in a separate variable because we are going to use it later when we perform our portfolio optimization.

In [22]:
```
h0 = universe['h.opt.previous']
```

Matrix of Risk Factor Exposures

Our dataframe contains several columns that we'll use as risk factors exposures. Extract these and put them into a matrix.

The data, such as industry category, are already one-hot encoded, but if this were not the case, then using `patsy.dmatrices` would help, as this function extracts categories and performs the one-hot encoding. We'll practice using this package, as you may find it useful with future data sets. You could also store the factors in a dataframe if you prefer.

How to use patsy.dmatrices

`patsy.dmatrices` takes in a formula and the dataframe. The formula tells the function which columns to take. The formula will look something like this:
`SpecRisk ~ 0 + USFASTD_AERODEF + USFASTD_AIRLINES + ...`
where the variable to the left of the ~ is the "dependent variable" and the others to the right are the independent variables (as if we were preparing data to be fit to a model).

This just means that the `pasty.dmatrices` function will return two matrix variables, one that contains the single column for the dependent variable `outcome`, and the independent variable columns are stored in a matrix `predictors`.

The `predictors` matrix will contain the matrix of risk factors, which is what we want. We don't actually need the `outcome` matrix; it's just created because that's the way patsy.dmatrices works.

In [23]:
```
formula = get_formula(risk_factors, "SpecRisk")
```

In [24]:
```
def model_matrix(formula, data):
    outcome, predictors = patsy.dmatrices(formula, data)
    return predictors
```

In [25]:
```
B = model_matrix(formula, universe)
BT = B.transpose()
```

Calculate Specific Variance

Notice that the specific risk data is in percent:

In [26]:
```
universe['SpecRisk'][0:2]
```

Out[26]:
```
0     9.014505
1    11.726327
Name: SpecRisk, dtype: float64
```

Therefore, in order to get the specific variance for each stock in the universe we first need to multiply these values by `0.01` and then square them:

In [27]:
```
specVar = (0.01 * universe['SpecRisk']) ** 2
```

Factor covariance matrix (TODO)

Note that we already have factor covariances from Barra data, which is stored in the variable `covariance`. `covariance` is a dictionary, where the key is each day's date, and the value is a dataframe containing the factor covariances.

In [28]:
```
covariance['20040102'].head()
```

Out[28]:

	Factor1	Factor2	VarCovar	DataDate
0	USFASTD_1DREVRSL	USFASTD_1DREVRSL	1.958869	20040102
1	USFASTD_1DREVRSL	USFASTD_BETA	1.602458	20040102
2	USFASTD_1DREVRSL	USFASTD_DIVYILD	-0.012642	20040102
3	USFASTD_1DREVRSL	USFASTD_DWNRISK	-0.064387	20040102
4	USFASTD_1DREVRSL	USFASTD_EARNQLTY	0.046573	20040102

In the code below, implement the function `diagonal_factor_cov` to create the factor covariance matrix. Note that the covariances are given in percentage units squared. Therefore you must re-scale them appropriately so that they're in decimals squared. Use the given `colnames` function to get the column names from `B`.

When creating factor covariance matrix, you can store the factor variances and covariances, or just store the factor variances. Try both, and see if you notice any differences.

In [29]:
```
def colnames(B):
    if type(B) == patsy.design_info.DesignMatrix:
        return B.design_info.column_names
    if type(B) == pandas.core.frame.DataFrame:
        return B.columns.tolist()
    return None
```

In [30]:
```
def diagonal_factor_cov(date, B):
    """
    Create the factor covariance matrix

    Parameters
    ----------
    date : string
          date. For example 20040102

    B : patsy.design_info.DesignMatrix OR pandas.core.frame.DataFrame
        Matrix of Risk Factors

    Returns
    -------
    Fm : Numpy ndarray
        factor covariance matrix
    """
    cv = covariance[date]
    k = np.shape(B)[1]
    Fm = np.zeros([k,k])
    for j in range(0,k):
        fac = colnames(B)[j]
        Fm[j,j] = (0.01**2) * cv.loc[(cv.Factor1==fac) & (cv.Factor2==fac
    return(Fm)

Fvar = diagonal_factor_cov(date, B)
```

Transaction Costs

To get the transaction cost, or slippage, we have to multiply the price change due to market impact by the amount of dollars traded:

$$ \mbox{tcost_{i,t}} = \% \Delta \mbox{price}_{i,t} \times \mbox{trade}_{i,t} $$
In summation notation it looks like this:
$$ \mbox{tcost}_{i,t} = \sum_i^{N} \lambda_{i,t} (h_{i,t} - h_{i,t-1})^2 $$
where $$ \lambda_{i,t} = \frac{1}{10\times \mbox{ADV}_{i,t}} $$

Note that since we're dividing by ADV, we'll want to handle cases when ADV is missing or zero. In those instances, we can set ADV to a small positive number, such as 10,000, which, in practice assumes that the stock is illiquid. In the code below if there is no volume information we assume the asset is illiquid.

In [31]:
```
def get_lambda(universe, composite_volume_column = 'ADTCA_30'):
    universe.loc[np.isnan(universe[composite_volume_column]), composite_v
    universe.loc[universe[composite_volume_column] == 0, composite_volume

    adv = universe[composite_volume_column]

    return 0.1 / adv

Lambda = get_lambda(universe)
```

Alpha Combination (TODO)

In the code below create a matrix of alpha factors and return it from the function `get_B_alpha`. Create this matrix in the same way you created the matrix of risk factors, i.e. using the `get_formula` and `model_matrix` functions we have defined above. Feel free to go back and look at the previous code.

In [32]:
```
def get_B_alpha(alpha_factors, universe):
    # TODO: Implement

    return model_matrix(get_formula(alpha_factors, "SpecRisk"), data = un

B_alpha = get_B_alpha(alpha_factors, universe)
```

Now that you have the matrix containing the alpha factors we will combine them by adding its rows. By doing this we will collapse the `B_alpha` matrix into a single alpha vector. We'll multiply by `1e-4` so that the expression of expected portfolio return, $\alpha^T \mathbf{h}$, is in dollar units.

In [33]:
```
def get_alpha_vec(B_alpha):
    """
    Create an alpha vecrtor

    Parameters
    ----------
    B_alpha : patsy.design_info.DesignMatrix
        Matrix of Alpha Factors

    Returns
    -------
    alpha_vec : patsy.design_info.DesignMatrix
        alpha vecrtor
    """

    # TODO: Implement
    scale = 1e-4
    alpha_vec = scale * np.sum(B_alpha, axis=1)
    return alpha_vec

alpha_vec = get_alpha_vec(B_alpha)
```

Optional Challenge

You can also try to a more sophisticated method of alpha combination, by choosing the holding for each alpha based on the same metric of its performance, such as the factor returns, or sharpe ratio. To make this more realistic, you can calculate a rolling average of the sharpe ratio, which is updated for each day. Remember to only use data that occurs prior to the date of each optimization, and not data that occurs in the future. Also, since factor returns and sharpe ratios may be negative, consider using a `max` function to give the holdings a lower bound of zero.

Objective function (TODO)

The objective function is given by:

$$ f(\mathbf{h}) = \frac{1}{2}\kappa \mathbf{h}_t^T\mathbf{Q}^T\mathbf{Q}\mathbf{h}_t + \frac{1}{2} \kappa \mathbf{h}_t^T \mathbf{S} \mathbf{h}_t - \mathbf{\alpha}^T \mathbf{h}_t + (\mathbf{h}_{t} - \mathbf{h}_{t-1})^T \mathbf{\Lambda} (\mathbf{h}_{t} - \mathbf{h}_{t-1}) $$

Where the terms correspond to: factor risk + idiosyncratic risk - expected portfolio return + transaction costs, respectively. We should also note that $\textbf{Q}^T\textbf{Q}$ is defined to be the same as $\textbf{BFB}^T$. Review the lessons if you need a refresher of how we get $\textbf{Q}$.

Our objective is to minimize this objective function. To do this, we will use Scipy's optimization function:

`scipy.optimize.fmin_l_bfgs_b(func, initial_guess, func_gradient)`

where:

- **func** : is the function we want to minimize
- **initial_guess** : is out initial guess
- **func_gradient** : is the gradient of the function we want to minimize

So, in order to use the `scipy.optimize.fmin_l_bfgs_b` function we first need to define its parameters.

In the code below implement the function `obj_func(h)` that corresponds to the objective function above that we want to minimize. We will set the risk aversion to be `1.0e-6`.

In [34]:
```
risk_aversion = 1.0e-6

def get_obj_func(h0, risk_aversion, Q, specVar, alpha_vec, Lambda):
    def obj_func(h):
        # TODO: Implement
        f = 0.0
        f += 0.5 * risk_aversion * np.sum( np.matmul(Q, h) ** 2 )
        f += 0.5 * risk_aversion * np.dot(h ** 2, specVar) #since Specifi
        f -= np.dot(h, alpha_vec)
        f += np.dot( (h - h0) ** 2, Lambda)

        return f

    return obj_func
```

Gradient (TODO)

Now that we can generate the objective function using `get_obj_func`, we can now create a similar function with its gradient. The reason we're interested in calculating the gradient is so that we can tell the optimizer in which direction, and how much, it should shift the portfolio holdings in order to improve the objective function (minimize variance, minimize transaction cost, and maximize expected portfolio return).

Before we implement the function we first need to know what the gradient looks like. The gradient, or derivative of the objective function, with respect to the portfolio holdings h, is given by:

$$ f'(\mathbf{h}) = \frac{1}{2}\kappa (2\mathbf{Q}^T\mathbf{Qh}) + \frac{1}{2}\kappa (2\mathbf{Sh}) - \mathbf{\alpha} + 2(\mathbf{h}_{t} - \mathbf{h}_{t-1}) \mathbf{\Lambda} $$
In the code below, implement the function `grad(h)` that corresponds to the function of the gradient given above.

In [35]:
```
def get_grad_func(h0, risk_aversion, Q, QT, specVar, alpha_vec, Lambda):
    def grad_func(h):
        # TODO: Implement
        g = risk_aversion * (np.matmul(QT, np.matmul(Q,h)) + (specVar * h
            - alpha_vec \
            + 2 * (h-h0) * Lambda

        return (np.asarray(g))

    return grad_func
```

Optimize (TODO)

Now that we can generate the objective function using `get_obj_func`, and its corresponding gradient using `get_grad_func` we are ready to minimize the objective function using Scipy's optimization function. For this, we will use out initial holdings as our `initial_guess` parameter.

In the cell below, implement the function `get_h_star` that optimizes the objective function. Use the objective function (`obj_func`) and gradient function (`grad_func`) provided within `get_h_star` to optimize the objective function using the `scipy.optimize.fmin_l_bfgs_b` function.

In [36]:
```
risk_aversion = 1.0e-6

Q = np.matmul(scipy.linalg.sqrtm(Fvar), BT)
QT = Q.transpose()

def get_h_star(risk_aversion, Q, QT, specVar, alpha_vec, h0, Lambda):
    """
    Optimize the objective function

    Parameters
    ----------
    risk_aversion : int or float
        Trader's risk aversion

    Q : patsy.design_info.DesignMatrix
        Q Matrix

    QT : patsy.design_info.DesignMatrix
        Transpose of the Q Matrix

    specVar: Pandas Series
        Specific Variance

    alpha_vec: patsy.design_info.DesignMatrix
        alpha vector

    h0 : Pandas Series
        initial holdings

    Lambda : Pandas Series
        Lambda

    Returns
    -------
    optimizer_result[0]: Numpy ndarray
        optimized holdings
    """
    obj_func = get_obj_func(h0, risk_aversion, Q, specVar, alpha_vec, Lam
    grad_func = get_grad_func(h0, risk_aversion, Q, QT, specVar, alpha_ve

    # TODO: Implement
    optimizer_result = scipy.optimize.fmin_l_bfgs_b(obj_func, h0, grad_fu

    return optimizer_result[0]

h_star = get_h_star(risk_aversion, Q, QT, specVar, alpha_vec, h0, Lambda)
```

After we have optimized our objective function we can now use, `h_star` to create our optimal portfolio:

In [37]:
```
opt_portfolio = pd.DataFrame(data = {"Barrid" : universe['Barrid'], "h.op
```

Risk Exposures (TODO)

We can also use `h_star` to calculate our portfolio's risk and alpha exposures.

In the cells below implement the functions `get_risk_exposures` and `get_portfolio_alpha_exposure` that calculate the portfolio's risk and alpha exposures, respectively.

In [38]:
```
def get_risk_exposures(B, BT, h_star):
    """
    Calculate portfolio's Risk Exposure

    Parameters
    ----------
    B : patsy.design_info.DesignMatrix
        Matrix of Risk Factors

    BT : patsy.design_info.DesignMatrix
        Transpose of Matrix of Risk Factors

    h_star: Numpy ndarray
        optimized holdings

    Returns
    --------
    risk_exposures : Pandas Series
        Risk Exposures
    """

    # TODO: Implement
    risk_exposures = np.matmul(B.T, h_star)

    return pd.Series(risk_exposures, index = colnames(B))

risk_exposures = get_risk_exposures(B, BT, h_star)
```

In [39]:
```
def get_portfolio_alpha_exposure(B_alpha, h_star):
    """
    Calculate portfolio's Alpha Exposure

    Parameters
    ----------
    B_alpha : patsy.design_info.DesignMatrix
        Matrix of Alpha Factors

    h_star: Numpy ndarray
        optimized holdings

    Returns
    --------
    alpha_exposures : Pandas Series
        Alpha Exposures
    """

    # TODO: Implement
    return pd.Series(np.matmul(B_alpha.transpose(), h_star), index = coln

portfolio_alpha_exposure = get_portfolio_alpha_exposure(B_alpha, h_star)
```

Transaction Costs (TODO)

We can also use `h_star` to calculate our total transaction costs: $$ \mbox{tcost} = \sum_i^{N} \lambda_{i} (h_{i,t} - h_{i,t-1})^2 $$

In the cell below, implement the function `get_total_transaction_costs` that calculates the total transaction costs according to the equation above:

In [40]:
```
def get_total_transaction_costs(h0, h_star, Lambda):
    """
    Calculate Total Transaction Costs

    Parameters
    ----------
    h0 : Pandas Series
        initial holdings (before optimization)

    h_star: Numpy ndarray
        optimized holdings

    Lambda : Pandas Series
        Lambda

    Returns
    -------
    total_transaction_costs : float
        Total Transaction Costs
    """

    # TODO: Implement
    # I had this formula wrong, so I checked the forums to get an idea on
    # https://hub.udacity.com/rooms/community:nd880:346730-project-581?co
    # contributions on April 9, 2019 by Ram Krishnan K. and Sarganil D.
    total_transaction_costs = np.sum((h_star-h0)**2 * Lambda)

    return total_transaction_costs

total_transaction_costs = get_total_transaction_costs(h0, h_star, Lambda)
```

Putting It All Together

We can now take all the above functions we created above and use them to create a single function, `form_optimal_portfolio` that returns the optimal portfolio, the risk and alpha exposures, and the total transactions costs.

In [41]:
```
def form_optimal_portfolio(df, previous, risk_aversion):
    df = df.merge(previous, how = 'left', on = 'Barrid')
    df = clean_nas(df)
    df.loc[df['SpecRisk'] == 0]['SpecRisk'] = median(df['SpecRisk'])

    universe = get_universe(df)
    date = str(int(universe['DataDate'][1]))

    all_factors = factors_from_names(list(universe))
    risk_factors = setdiff(all_factors, alpha_factors)

    h0 = universe['h.opt.previous']

    B = model_matrix(get_formula(risk_factors, "SpecRisk"), universe)
    BT = B.transpose()

    specVar = (0.01 * universe['SpecRisk']) ** 2
    Fvar = diagonal_factor_cov(date, B)

    Lambda = get_lambda(universe)
    B_alpha = get_B_alpha(alpha_factors, universe)
    alpha_vec = get_alpha_vec(B_alpha)

    Q = np.matmul(scipy.linalg.sqrtm(Fvar), BT)
    QT = Q.transpose()

    h_star = get_h_star(risk_aversion, Q, QT, specVar, alpha_vec, h0, Lam
    opt_portfolio = pd.DataFrame(data = {"Barrid" : universe['Barrid'], "

    risk_exposures = get_risk_exposures(B, BT, h_star)
    portfolio_alpha_exposure = get_portfolio_alpha_exposure(B_alpha, h_st
    total_transaction_costs = get_total_transaction_costs(h0, h_star, Lam

    return {
        "opt.portfolio" : opt_portfolio,
        "risk.exposures" : risk_exposures,
        "alpha.exposures" : portfolio_alpha_exposure,
        "total.cost" : total_transaction_costs}
```

Build tradelist

The trade list is the most recent optimal asset holdings minus the previous day's optimal holdings.

In [42]:
```
def build_tradelist(prev_holdings, opt_result):
    tmp = prev_holdings.merge(opt_result['opt.portfolio'], how='outer', o
    tmp['h.opt.previous'] = np.nan_to_num(tmp['h.opt.previous'])
    tmp['h.opt'] = np.nan_to_num(tmp['h.opt'])
    return tmp
```

Save optimal holdings as previous optimal holdings.

As we walk through each day, we'll re-use the column for previous holdings by storing the "current" optimal holdings as the "previous" optimal holdings.

In [43]:
```
def convert_to_previous(result):
    prev = result['opt.portfolio']
    prev = prev.rename(index=str, columns={"h.opt": "h.opt.previous"}, co
    return prev
```

Run the backtest

Walk through each day, calculating the optimal portfolio holdings and trade list. This may take some time, but should finish sooner if you've chosen all the optimizations you learned in the lessons.

In [44]:
```
trades = {}
port = {}

for dt in tqdm(my_dates, desc='Optimizing Portfolio', unit='day'):
    date = dt.strftime('%Y%m%d')

    result = form_optimal_portfolio(frames[date], previous_holdings, risk_
    trades[date] = build_tradelist(previous_holdings, result)
    port[date] = result
    previous_holdings = convert_to_previous(result)
```

```
Optimizing Portfolio: 100%|██████████| 252/252 [22:32<00:00,  5.37s/day]
```

Profit-and-Loss (PnL) attribution (TODO)

Profit and Loss is the aggregate realized daily returns of the assets, weighted by the optimal portfolio holdings chosen, and summed up to get the portfolio's profit and loss.

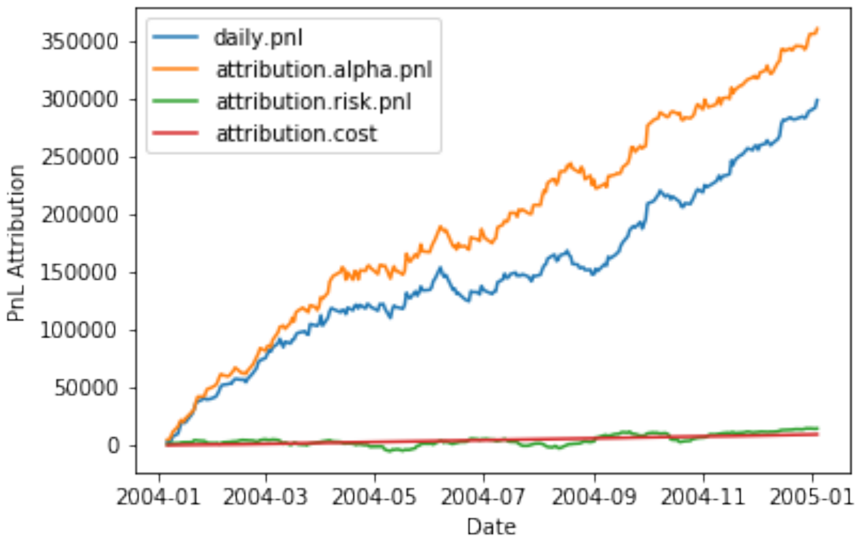
The PnL attributed to the alpha factors equals the factor returns times factor exposures for the alpha factors.

$$ \mbox{PnL}_{alpha}= f \times b_{alpha} $$
Similarly, the PnL attributed to the risk factors equals the factor returns times factor exposures of the risk factors.

$$ \mbox{PnL}_{risk} = f \times b_{risk} $$
In the code below, in the function `build_pnl_attribution` calculate the PnL attributed to the alpha factors, the PnL attributed to the risk factors, and attribution to cost.

In [45]:
```
## assumes v, w are pandas Series
def partial_dot_product(v, w):
    common = v.index.intersection(w.index)
    return np.sum(v[common] * w[common])

def build_pnl_attribution():

    df = pd.DataFrame(index = my_dates)

    for dt in my_dates:
        date = dt.strftime('%Y%m%d')

        p = port[date]
        fr = facret[date]

        mf = p['opt.portfolio'].merge(frames[date], how = 'left', on = "B

        mf['DlyReturn'] = wins(mf['DlyReturn'], -0.5, 0.5)
        df.at[dt,"daily.pnl"] = np.sum(mf['h.opt'] * mf['DlyReturn'])

        # TODO: Implement
        # I was completely lost here, and it took me a while to find a so
        # of some Forum discussions:
        # https://hub.udacity.com/rooms/community:nd880:346730-project-58

        df.at[dt,"attribution.alpha.pnl"] = partial_dot_product(fr, p['al
        df.at[dt,"attribution.risk.pnl"] = partial_dot_product(fr, p['ris
        df.at[dt,"attribution.cost"] = p['total.cost']

    return df
```

In [49]:
```
attr = build_pnl_attribution()

for column in attr.columns:
        plt.plot(attr[column].cumsum(), label=column)
plt.legend(loc='upper left')
plt.xlabel('Date')
plt.ylabel('PnL Attribution')
plt.show()
```



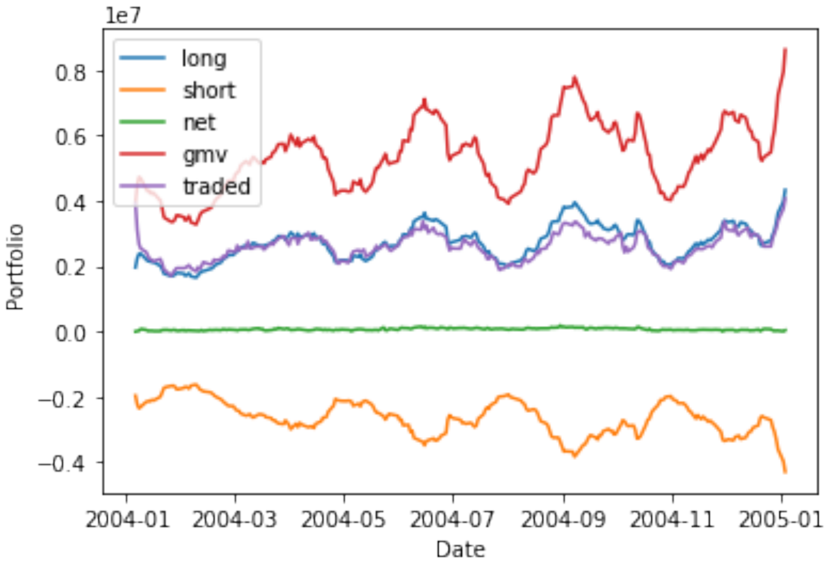
Build portfolio characteristics (TODO)

Calculate the sum of long positions, short positions, net positions, gross market value, and amount of dollars traded.

In the code below, in the function `build_portfolio_characteristics` calculate the sum of long positions, short positions, net positions, gross market value, and amount of dollars traded.

In [50]:
```
def build_portfolio_characteristics():
    df = pd.DataFrame(index = my_dates)

    for dt in my_dates:
        date = dt.strftime('%Y%m%d')

        p = port[date]

        tradelist = trades[date]

        h = p['opt.portfolio']['h.opt']

        # TODO: Implement
        # Again, I found this discussion from the Forums helpful:
        # https://hub.udacity.com/rooms/community:nd880:346730-project-58

        long = np.sum([item for item in h if item >= 0])
        short = np.sum([item for item in h if item < 0])

        df.at[dt,"long"] = long
        df.at[dt,"short"] = short
        df.at[dt,"net"] = long + short
        df.at[dt,"gmv"] = abs(long) + abs(short)
        df.at[dt,"traded"] = np.sum(abs(tradelist['h.opt'] - tradelist['h

    return df
```

In [51]:
```
pchar = build_portfolio_characteristics()

for column in pchar.columns:
        plt.plot(pchar[column], label=column)
plt.legend(loc='upper left')
plt.xlabel('Date')
plt.ylabel('Portfolio')
plt.show()
```



Optional

Choose additional metrics to evaluate your portfolio.

In [ ]:
```
# Optional
```

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.