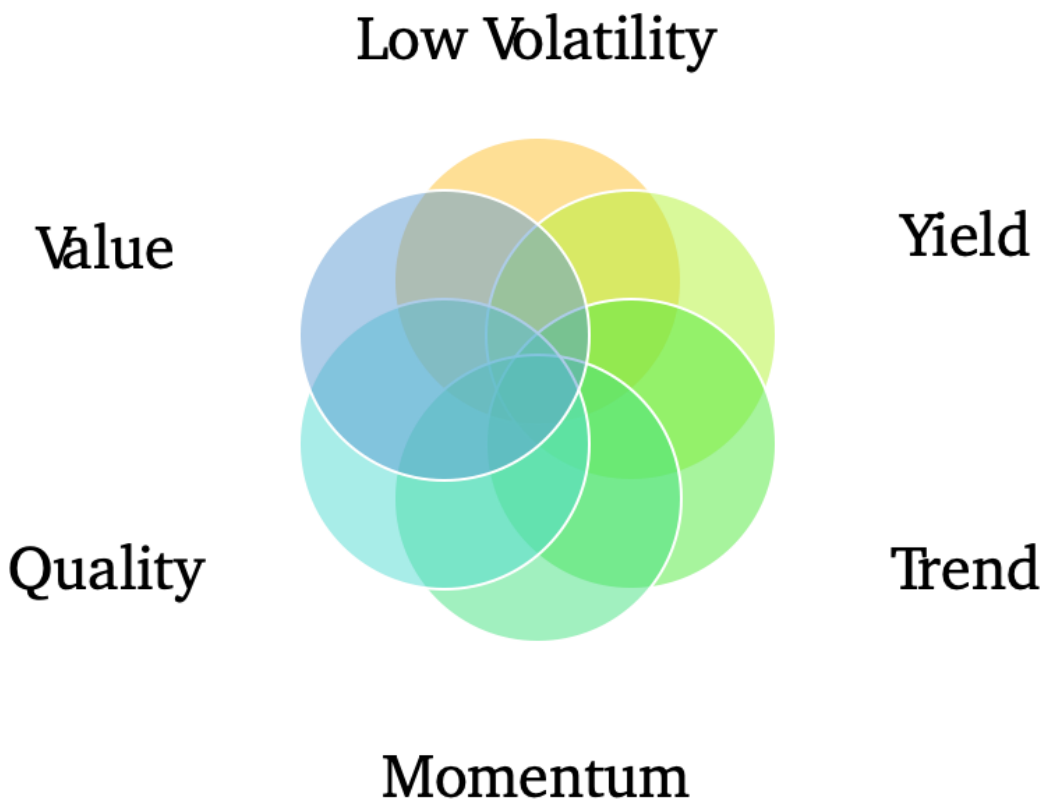
How to Build a Multi-Factor Equity Portfolio in Python

Steven Downey, CFA, CMT Follow
Jun 10, 2020 · 12 min read ★



Summary

- **Multi-factor portfolios combine different investment characteristics, such as value and momentum, into a single portfolio as a way to reap the risk/behavior premium associated with different historically statistically significant investment styles**
- **It can be simple to create a single factor portfolio, such as high book-to-market, which would be considered a value portfolio, but combining multiple factors through a composite of measures is slightly more sophisticated**
- **Using USA equity price and fundamental data, we can construct a multi-factor portfolio that aims to capture the low-volatility, quality, momentum, trend, and value factors**
- **You will leave with a more nuanced understanding of multi-factor portfolio construction and code to backtest and research yourself.**

Investment factors are a relatively new label for long-standing investment strategies executed by a variety of successful traders and investors. Jesse Livermore exemplified a trend-following investment style ("let your winners run and cut your losers short"), Warren Buffet has exemplified a Value and Quality style with the use of leverage ("buy good companies at a fair price"), and George Soros, in part, executed a momentum/trend-following investment strategy.

Since the Fama French paper on the Value and Size factor, academic research, and subsequent practitioner implementation, has abounded in factor research.

As a practitioner myself, I sense right now multi-factor investing has been all the rage for the last 5 years or so, with ESG being the current love affair. I was curious about how to go about building a multi-factor portfolio, and there was a lot written on the concepts but little on the nuts and bolts of the coding aspect. So a few months ago, I decided to give it a go.

Methodology

In the literature, there are two different ways to build a multi-factor portfolio, bottom-up or top-down. Top-down combines targeted factor portfolios, think "combining silos together", and bottom-up ranks each security on their overall factor rank and chooses

the securities that have the best overall score of all the factors. All of the literature I have come across (here, and here)favors the bottom-up approach (except this one from AlphaArchitect on combining momentum and value separately), in part due to the fact you may duplicate trades in the top-down approach and potentially have better diversification with a blended signal.

I decided to use a bottom-up blended signal approach in building the Python script, with the aim of gaining exposure to the following factors:

- Momentum
- Trend
- Shareholder Yield
- Value
- Quality
- Low Volatility

Most of the multi-factor methodologies I have come across tend to leave out the Trend factor, even though there is a lot of research backing its statistical significance (here and the superb trend-following textbook *Trend Following with Managed Futures: The Search for Crisis Alpha)*.

I used US equity pricing and fundamental data from Sharadar, which will cost you depending on the size of your firm, and if you are a finance professional (I currently pay around $3,000 a year). I believe it is cheaper if you don't work at a financial services firm but you will need to find that out yourself. It is the best value for the price I have found when it comes to complete, listed and delisted, US equity price and fundamental data going back to around 1999. Having said that, if price is an issue for you, you could use pure price data and construct a portfolio with non-fundamental factors, such as momentum, trend, and low volatility, though you will want to get listed and delisted prices to eliminate survivorship bias.

There are many different ways to construct a multi-factor portfolio:

- Vanguard uses an initial screen to weed out the most volatile equities as a way of capturing the low volatility premium
- MSCI builds its multi-factor index while trying to constrain turnover and sector allocation, among other nuances
- As a programmer, there are different ways to code the system. I will be using for loops as it is intuitive to me and due to my level of programming skill (medium), but you could potentially use object-oriented programming to build the system similar to the Quantopian infrastructure.

Questions you may want to explore when building your model:

- How often should I rebalance?
- What factors should I include and how will I measure them?
- How will I weight the chosen securities? Should I use equal weight or something more complex like equal risk contribution?
- Should I have any turnover or sector exposure constraints?

You can access the full python code on GitHub, but I will try to explain it step by step here.

So let's load the necessary libraries. I have created a perfomance_analysis python file that contains easy to use performance metric functions that are also available on GitHub.

```
import os
import datetime
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from pandas_datareader.famafrench import get_available_datasets
import pandas_datareader.data as web

#use the performance_analysis python file to import functions
os.chdir('/PUT FILE DESTINATION HERE')
```

```
from performance_analysis import annualized_return
from performance_analysis import annualized_standard_deviation
from performance_analysis import max_drawdown
from performance_analysis import gain_to_pain_ratio
from performance_analysis import calmar_ratio
from performance_analysis import sharpe_ratio
from performance_analysis import sortino_ratio

pd.set_option('display.max_columns', 200)
pd.set_option('display.max_rows', 1000)
plt.style.use('ggplot')
#turn off pandas warning for index/slicing as copy warning
pd.options.mode.chained_assignment = None  # default='warn'
```

We need to load the data and filter out the data frame to just focus on US-listed companies.

```
#%%
############Fundamental and Equity Prices#############################

#fundamental data
fundamental_data = (
    pd.read_csv('FUNDAMENTAL_FILE.csv')
    )

#import all of the equity price data from csv from Sharadar
equity_prices = (
    pd.read_csv('PRICE FILE.csv')
    )

#get ticker meta data
tickers_df = (
    pd.read_csv('TICKER INFO.csv', low_memory=False)
    )

#filter out companies not based in USA
tickers_df1 = tickers_df[tickers_df['location'].notnull()]
tickers_df1 =
tickers_df1[tickers_df1['location'].str.contains("U.S.")]

#select needed columns to filter out sector in fundamental
tickers_df1 = (
    tickers_df1[['ticker', 'sector', 'name',
                 'industry', 'scalemarketcap']]
    )

#create set and list of all tickers
myset_ticker = set(tickers_df1.ticker)
```

```
list_tickers = list(myset_ticker)

#filtered USA fundamental data
USA_fundamentals =
fundamental_data[fundamental_data['ticker'].isin(list_tickers)]
#%%
```

So we have all of the data on US-listed and delisted companies but we may want to filter out a specific sector or include the whole market.

```
#%%

##########     Filtering the Dataset        ####################

#test_sector = 'Technology' #

#### The 11 Sectors you can choose from ####

#'Healthcare', 'Basic Materials', 'Financial Services',
#'Technology', 'Industrials', 'Consumer Cyclical', 'Real Estate',
#'Consumer Defensive', 'Communication Services', 'Energy',
#'Utilities'

#If you want to just test the sector
#sector_stocks = tickers_df1[tickers_df1['sector'] == test_sector]

#OR

#If you wanted to remove Real estate and Financial Services
#sector_stocks = tickers_df1[tickers_df1['sector'] != 'Real Estate']
#sector_stocks = sector_stocks[sector_stocks['sector'] != 'Financial
Services']

#OR

#To test all the market
sector_stocks = tickers_df1
```

We are going to put the tickers in a list and only keep the Annual Reported Twelve months trailing data. We also want to keep the earliest data point whenever there were revisions in the data. In this example, I filtered out companies that had less than a $1 billion market cap to ensure liquidity when executing the trades.

```
#put tickers to list from sector specified
sector_tickers = sector_stocks['ticker'].tolist()

#fundamentals imported already
fundamentals =
USA_fundamentals[USA_fundamentals.ticker.isin(sector_tickers)]

#Choose dimension rolling 'twelve month as reported' 'ART'. Sharadar
has revisions
#and that would be lookahead bias to use that data.
fundamentals = fundamentals[fundamentals.dimension == 'ART']

#Find data rows where fundamentals have been restated for previous
quarter
#and we want to remove for backtesting since at the time you only
have the first
#release of data and not subsequent revisions
duplicateRowsDF = fundamentals[fundamentals.duplicated(['ticker',
'calendardate'])]

print("Duplicate Rows based on 2 columns are:", duplicateRowsDF,
sep='\n')

fundamentals = fundamentals.drop_duplicates(subset = ['ticker',
'calendardate'],\
                                            keep = 'first')

duplicateRowsDF = fundamentals[fundamentals.duplicated(['ticker',
'calendardate'])]

#make sure there are no duplicates
print("Duplicate Rows based on 2 columns are:", duplicateRowsDF,
sep='\n')

#filter out companies with less than $1 billion market cap or another
market cap
#that suits your fancy
Data_for_Portfolio = fundamentals[fundamentals['marketcap'] >= 1e9]

#put tickers in a list
tickers = Data_for_Portfolio['ticker'].tolist()
print('There are ' + str(len(set(tickers))) + ' tickers') #number of
unique tickers

There are 4780 tickers
```

So we have 4780 tickers or stocks over the life of the dataset with the given filters

We are going to take the sector info and combine it with the Fundamental data frame.

```
#### Map Sector info onto the Fundamental DataFrame to use later ###

#create the dictionary with values and keys as dates
keys = tickers_df1['ticker']
values = tickers_df1['sector']
Dictionary_Sector_values = dict(zip(keys, values))

Data_for_Portfolio['sector'] = Data_for_Portfolio\
    ['ticker'].map(Dictionary_Sector_values)
```

Now here is the fun part — Creating the metrics to give us Factor Scores. Below I am creating a few ratios that combined will give each factor score. For example, with the value factor, I am creating an earnings/price, EBITDA/EV, and FCF/price ratio that will be used later to create the final value factor score. You can and should adjust these if you feel there is a better and more robust way to create a factor score.

```
### Value Factor ###
Data_for_Portfolio['E/P'] = Data_for_Portfolio['netinc'] / \
    Data_for_Portfolio['marketcap']
Data_for_Portfolio['EBITDA/EV'] = Data_for_Portfolio['ebitda'] / \
    Data_for_Portfolio['ev']
Data_for_Portfolio['FCF/P'] = Data_for_Portfolio['fcf'] / \
    Data_for_Portfolio['marketcap']

### Shareholder Yield ###
Data_for_Portfolio['Shareholder Yield'] = \
    -((Data_for_Portfolio['ncfdebt'] + \
       Data_for_Portfolio['ncfdiv'] + \
           Data_for_Portfolio['ncfcommon']) /
Data_for_Portfolio['marketcap'])

### Quality Factor - ideas taken from Alpha Architect QV model ####

####Long Term Business Strength

#Can you generate free cash flow?
Data_for_Portfolio['FCF/Assets'] = Data_for_Portfolio['fcf'] / \
    Data_for_Portfolio['assets']

#Can you generate returns on investment?
Data_for_Portfolio['ROA'] = Data_for_Portfolio['roa']
Data_for_Portfolio['ROIC'] = Data_for_Portfolio['roic']

#Do you have a defendable business model?
Data_for_Portfolio['GROSS MARGIN'] =
Data_for_Portfolio['grossmargin']
```

```
#Current Financial Strength

Data_for_Portfolio['CURRENT RATIO'] =
Data_for_Portfolio['currentratio']
Data_for_Portfolio['INTEREST/EBITDA'] = Data_for_Portfolio['intexp']
/ \Data_for_Portfolio['ebitda']
```

Preparing the For Loop for Portfolio Implementation

We need to prepare the dates, variables, and data frames required to use the for loop.

We want to slice the time series data into In Sample and Out of Sample data if we are genuinely curious about real-world implementation vs. backtesting a million times to find the best historical fit. In this example, the In Sample will be September 30, 2000, to September 30, 2012. Out of Sample will be all the data after that. This python file is only In Sample but on GitHub there is an Out of Sample file that has everything the same except the dates.

```
t0 = time.time() #I like to time my code to see its speed

#sort out Sector Prices
Sector_stock_prices = equity_prices.loc  \
    [equity_prices['ticker'].isin(tickers)]

Data_for_Portfolio = Data_for_Portfolio.dropna()

#Using the same in sample dates here and for equal weight benchmark

f_date = datetime.date(2000, 9, 30)
l_date = datetime.date(2012, 9, 30) #choosing the last date, results
in last
#date for returns is l_date + 1 quarter

delta = l_date - f_date
quarters_delta = np.floor(delta.days/(365/4))
quarters_delta = int(quarters_delta)
first_quarter = str('2000-09-30') #using f_date
Data_for_Portfolio_master = pd.DataFrame(Data_for_Portfolio)
```

You will need to decide if you want to create a decile/quintile portfolio, or a top 5 company portfolio, etc. In this example, I will choose the company with the highest factor loading from each sector (11 sectors) and short the lowest factor loading from

each sector. This strategy has its obvious drawbacks versus a decile portfolio, which would have around 40–90 companies, but it is easier to implement the trade execution manually.

I chose to winsorize at the 2.5% level so that the top and bottom 2.5% of each metric is compressed to minimize the effect of outliers.

```
#choose if you want percentiles or fixed number of companies in long
portfolio

Percentile_split = .1

#OR
Companies_in_Portfolio = 5

Winsorize_Threshold = .025 #used to determine the winsorize level.

Portfolio_Turnover = pd.DataFrame()
portfolio_returns = pd.DataFrame()

#extracting and sorting the price index from the stock price df for
#use in the for loop

price_index = Sector_stock_prices.set_index('date')
price_index = price_index.index
price_index = price_index.unique()
price_index = pd.to_datetime(price_index)
price_index = price_index.sort_values()
```

Running the For Loop and Portfolio

It will be easier to read the for loop directly from the python file vs. here in Medium.

From a high level, we are going to do the following:

- Slice the data to look at the initial date quarter and the associated trailing twelve-month fundamental data
- Create the Value Factor, Quality Factor, Shareholder Yield Factor, and Low Volatility Factor scores using their respective Z score to normalize the results

- Take the equities with fundamental data and then create their respective Trend and Momentum Factor scores
- Combine the Factor Scores to arrive at a Total Score. In my example, the Total Score sums the first four factors and then multiplies by the sum of the Momentum and Trend Factors. Essentially, I only want to invest in cheap, quality, high yield, low volatility companies that also have high momentum and positive trend.
- I then filter out the highest factor loading equities and the worst for each sector
- I then get the price data and execute the trades for the end of the next quarter to avoid lookahead bias. Remember, companies will report their Q1 earnings sometime in the middle of Q2.
- I do the same when creating the lowest factor loading equities
- Finally, I combine the data and create a Long/Short portfolio
- Rinse and repeat every 4 quarters to create an annual rebalance schedule. You could obviously change the frequency of rebalancing, but you want to hold the securities for the duration it takes to capture the respective factor premium (i.e. momentum a few months, value a year or longer)

Two things to note:

- to create a more robust portfolio you can create 4 different portfolios that rebalance every 4 quarters but do so on successive quarters, resulting in a combined overall portfolio that is not as sensitive to the rebalance dates and minimizes timing luck.
- Running the for loop took my medium quality MacBook (2.9 GHz Dual-Core Intel Core i5, 16 GB memory) 3 minutes to run.

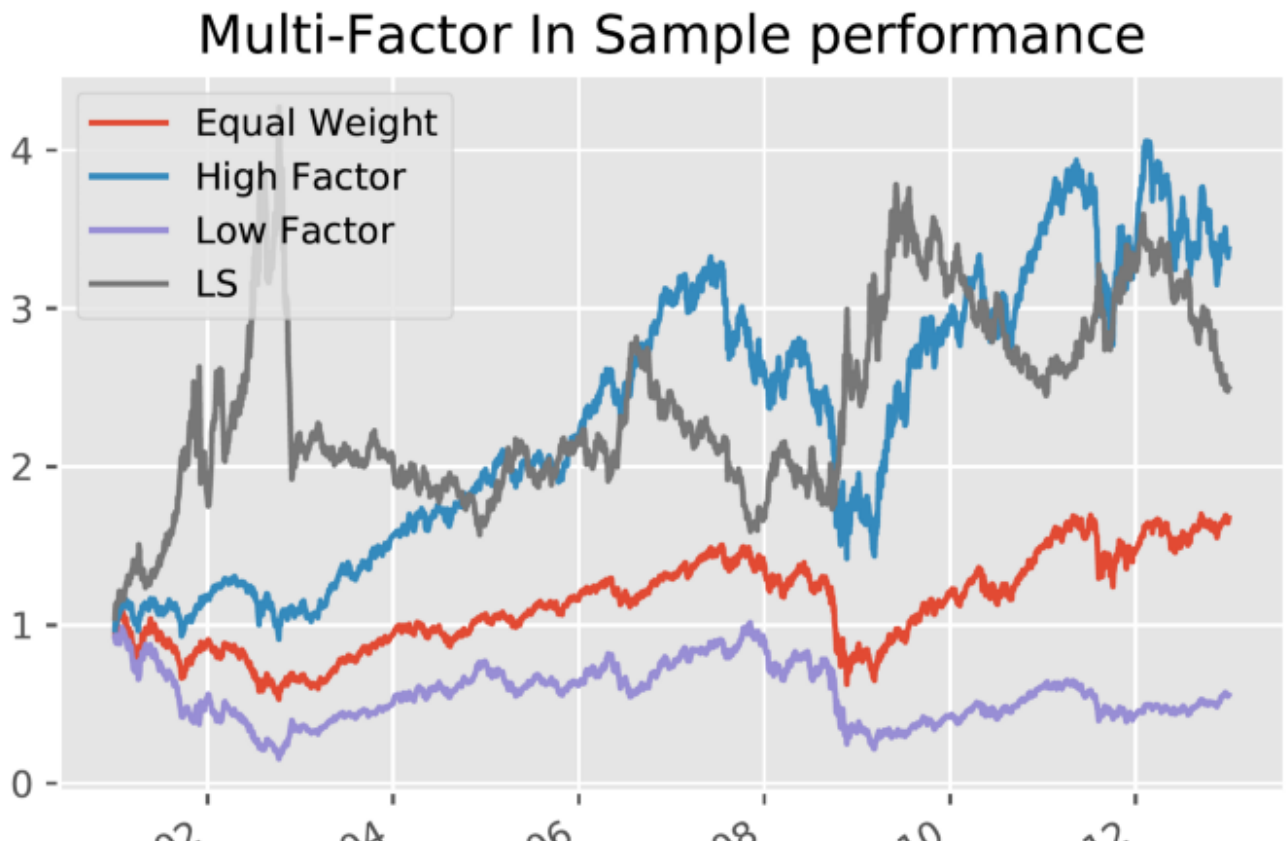
I also create an equal weight benchmark to compare risk and performance.

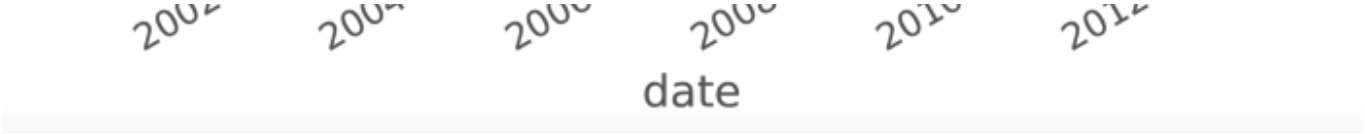
Below is the resulting top factor stocks from each sector for the initial quarter used, for example. There are really 150 columns in this dataframe but it would be hard to view here.

	ticker	dimension	calendardate	Trend Score	Momentum Score	Total Score
442	UST1	ART	2000-09-30	1.00	-0.158141	13.277910
307	NLOK	ART	2000-09-30	0.00	-0.670522	12.182058
411	SXCL	ART	2000-09-30	0.00	-1.229010	10.576490
462	WDR	ART	2000-09-30	1.00	0.988937	10.518097
174	EYE1	ART	2000-09-30	0.00	-1.180301	8.975452
306	NL	ART	2000-09-30	1.00	0.467530	8.613415
79	CBT	ART	2000-09-30	0.25	0.062802	8.290021
245	KRI	ART	2000-09-30	1.00	-0.285140	8.040494
138	DQE	ART	2000-09-30	0.00	-0.366047	7.077572
416	TDS	ART	2000-09-30	0.00	-0.495116	5.728254
247	LAMR	ART	2000-09-30	0.00	-0.689690	-4.263156

Simulated Performance

How did the In Sample portfolio do? The high factor portfolio seemed to outperform relative to the low factor portfolio following the recessions in early 2000 and 2008–2009. Note, that many of the factors in the portfolio, such as quality, low volatility, and trend, did not have a lot of academic research published on them during this time period, so you have to be skeptical of whether you would have thought to actually implement this strategy in real-time.





Risk and Returns

The following simply gets the risk free rate from the Kenneth French data library and then computes specific risk and return measures.

```
len(get_available_datasets())

ds = web.DataReader('F-F_Research_Data_Factors_daily', 'famafrench',
start='1990-08-30')

print(ds['DESCR'])

ds[0].head()

data = ds[0]
data = data.dropna()
data = data/100 #convert to percent returns
RF_data = (1+data['RF']).cumprod()

RF_start_date = portfolio_index.first_valid_index()
RF_end_date = portfolio_index.last_valid_index()

RF_data = pd.DataFrame(RF_data[RF_start_date:RF_end_date])

#########Calculate Risk and Performance#############################
annualized_return(RF_data)
RF_Ann_Return_df = annualized_return(RF_data)
RF_Ann_Return = np.round(float(RF_Ann_Return_df.iloc[:, 1]), 4)

sum(portfolio_returns['LS'])/(portfolio_returns.shape[0]/252)

returns = annualized_return(portfolio_index)
Stddev = annualized_standard_deviation(portfolio_index)
Sector_Perf = returns.merge(Stddev)

Sharpe_Ratios = sharpe_ratio(portfolio_index, RF_Ann_Return)
Sector_Perf = Sector_Perf.merge(Sharpe_Ratios)

Sortino_Ratios = sortino_ratio(portfolio_index, RF_Ann_Return)
Sector_Perf = Sector_Perf.merge(Sortino_Ratios)

Max_DD = max_drawdown(portfolio_index)
```

```
Sector_Perf = Sector_Perf.merge(Max_DD)

Calmar_Ratios = calmar_ratio(portfolio_index)
Sector_Perf = Sector_Perf.merge(Calmar_Ratios)

Gain_To_Pain = gain_to_pain_ratio(portfolio_index)
Sector_Perf = Sector_Perf.merge(Gain_To_Pain)
print(Sector_Perf)
```

Here is the combined statistics

```
                     Annualized Return  Standard Deviation
Portfolio
Equal Weight           0.049621           0.248990
High Factor            0.109952           0.253698
Low Factor            -0.041586           0.380391
LS                     0.075775           0.258370

                  Max Drawdown  Calmar Ratio    Gain to Pain Ratio
Portfolio
Equal Weight     -0.584042      0.084961           1.059211
High Factor      -0.572805      0.191954           1.102631
Low Factor       -0.843569     -0.049298           1.014555
LS               -0.632328      0.119835           1.078339

                       Sharpe Ratio (RF = 0.0183)   Sortino Ratio
Portfolio
Equal Weight             0.125792                    0.179931
High Factor              0.361265                    0.500191
Low Factor              -0.157434                   -0.216125
LS                       0.222451                    0.315889
```

Statistical Significance

Below is the code to see if the Long/Short portfolio return is statistically different than 0.

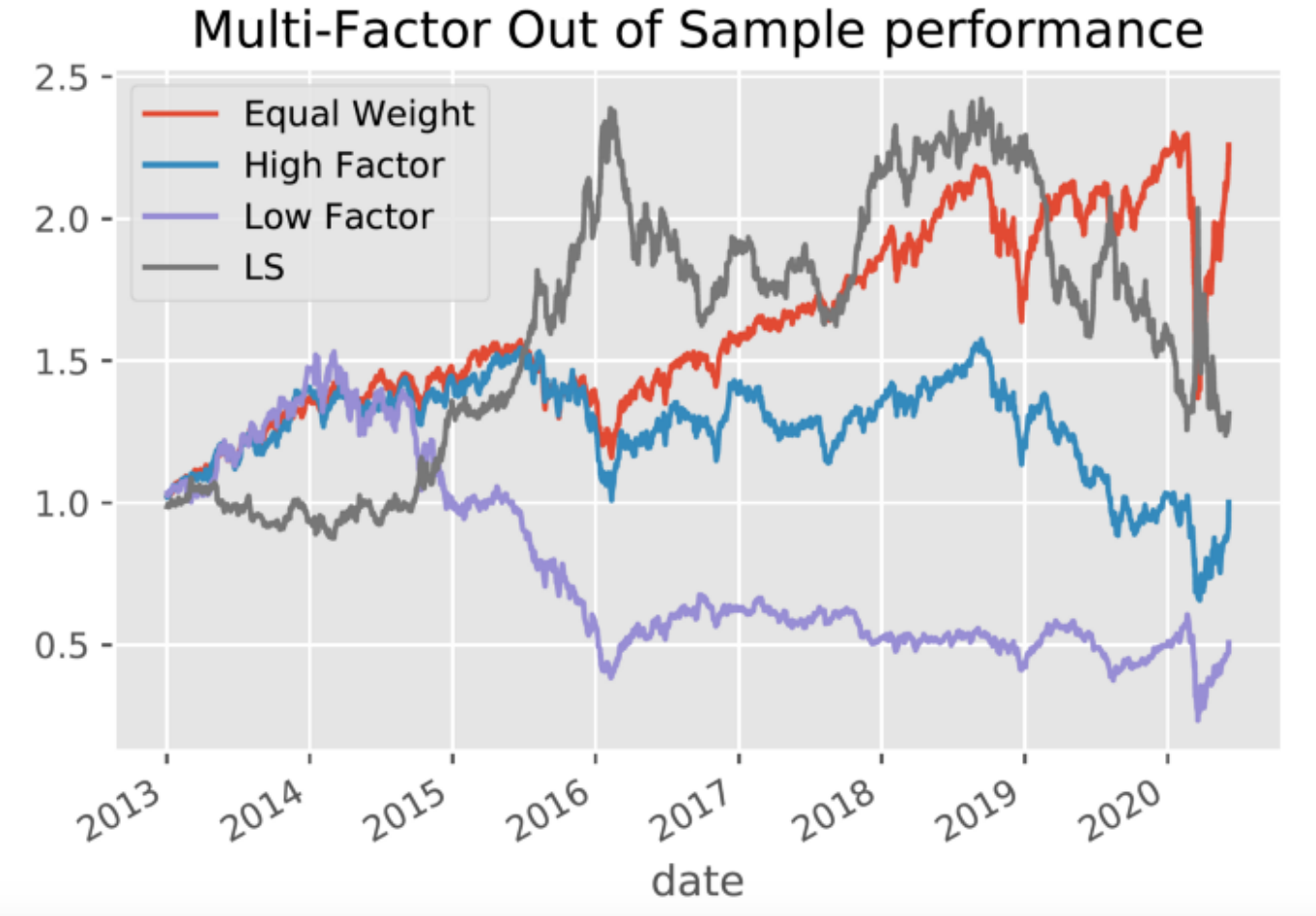
```
#####Testing Statistical Significance of L/S Portfolio#########
t2, p2 = stats.ttest_ind(Combined_Portfolio_returns['LS'], ZEROS)
print("t = " + str(t2))
print("p = " + str(p2/2)) #one sided t test

t = 1.473130478466982
p = 0.07038402579342423
```

So even though the Long/Short portfolio had an annualized geometric return of 7.5%, it was not statistically significant at the 5% level.

Out of Sample — The Real Test

So how did this strategy hold up on out of sample data?



Observations:

- Avoiding the Low Factor stocks was a wise choice, with massive drawdowns from 2014 to the end of 2016.
- The High Factor stocks had an average performance relative to the Equal Weight portfolio up until 2016, which then saw a large divergence.
- I estimate that the large relative negative performance compared to the Equal Weight portfolio is due to the high concentration of the Factor portfolios. A decile

portfolio would potentially fix this, but we can only run the out of sample test once to maintain integrity.

Conclusion

- Even if an in sample strategy has positive performance, it doesn't mean it will be statistically significant or perform well out of sample.
- There are a million ways to measure factors, choose which factors to include, and how to combine them to give you a resulting portfolio. Choose wisely!
- Even with a systematic investment strategy, how the strategy is constructed and implemented requires a lot of discernment, creativity, and discretion.
