# Financial Data Science Optimization

PS APAN 5440 Financial Data Science and Machine Learning
Final Report
Professor William Ritter
Fall 2020

Zian Gong
Jiaran Yu
Shuyang Li
Minjian Hu
Kexin Lian

## Introduction

For the final project, our group uses Python to build a fairly realistic back-tester on the given trading data set, which will perform portfolio optimization that uses APT factors. We also build a performance attribution to show the top drivers of the portfolio return.

The report will be divided into three parts: preprocess, optimization and backtest. The preprocess aims to clean and organize the data and make the following backtesting process viable. Checkpoint 2 to 4 takes steps to analyze the trading dataset and select multiple factors that we need for portfolio optimization. Python packages that we mainly use include Sys, Pandas, NumPy, SciPy and we also visualized our results using matplotlib.

## Preprocess

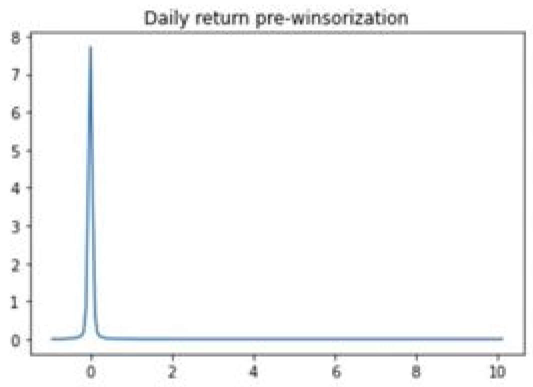
### Clean and Sort

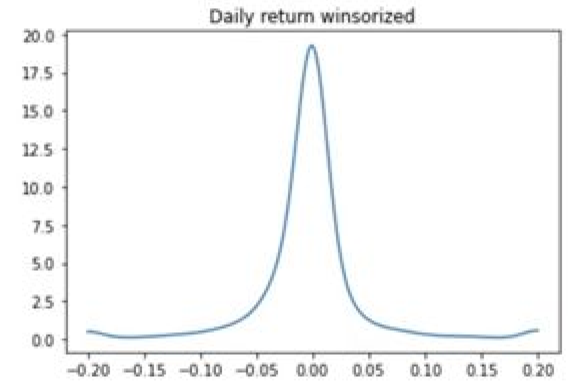
After loading all necessary libraries and pickle files, to prevent invalid syntax, we changed one of the column name, from "1DREVRSL" to "DREVRSL".

Then we define a function called sort_cols, which helps to put columns in alphabetical order for the convenience of future analysis.

Next, we set the minimum and maximum range by defining the function “wins”. The wins function takes a numpy array, and floating-point numbers denoting lower and upper bounds as inputs, then outputs a winsorized numpy array. The function performs a process called “Winsorizing” that helps to avoid extreme positive and negative values.

After that, we define a function called “clean_nas”, which can replace NaN with zero for certain columns. Finally, we define the function “density_plot”, which takes a data vector and a title for the plot, and produces a kernel density plot. The outcome is shown below.





## Estimate factor returns

We first define a function called get_formula which analyzes the relationships between independent and dependent variables. In our case, we were making the linear form of the APT model. And our dependent variable is called "Ret", which refers to the daily return.

Then we define another function, factors_from_names to extract the factor names from the column lists.

Next, we created a function "estimate_factor_returns", which takes the data frame corresponding to the model data for one particular day as input, and outputs the factor returns. Here, as a filter, we only choose the stocks with A IssuerMarketCap greater than 1 billion dollars and winsorize the returns for fitting based on an interval between -0.25 and 0.25. We remove the null values to prevent dropping rows with missing values, which is the default missing handling.

After collecting all these coefficients into our new data frame facret, it's time to choose our alpha factors. Here, we have tried different combinations of linear regression factors, including:

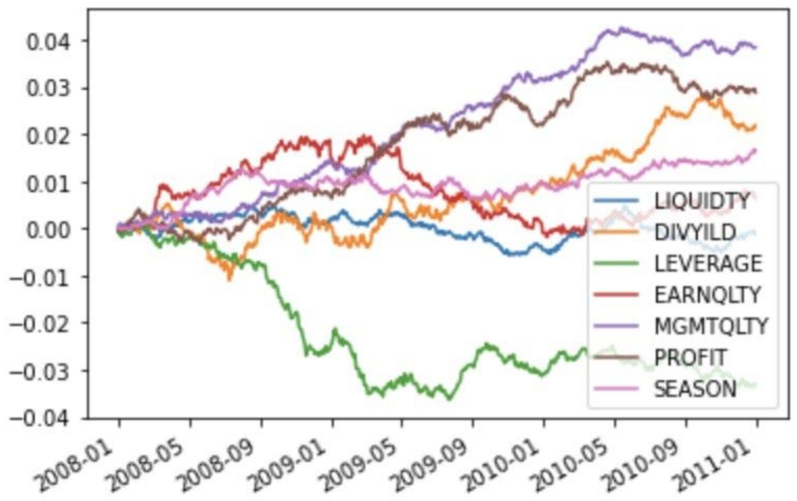
1. [ "DREVERSAL", "EARNYIELD", "VALUE", "SENTIMENT" ] -------Commonly used Barra US Equity Risk Model

2. **['LIQUIDTY','DIVYILD','LEVERAGE','EARNQLTY','MGMTQLTY','PROFIT', 'SEASON']**--------Is built Based on our financial and nonfinancial considerations. Financial metrics include profitability (earn quality, profit, dividend yield), liquidity, and debt level (leverage); Non-financial metrics includes management quality. And considering some special industry, we also include seasonality.

3. [ "DREVERSAL", "EARNYIELD", "VALUE", "SENTIMENT", "LEVERAGE" , "GROWTH" ]

By comparing the efficiency and performance of backtesting, we finally choose the second combination. And plot the cumulative summations of factor returns for the chosen alpha factors.

The output graph of the linear model is shown below.



## Optimization

In this part, we are trying to prepare for generating the optimal portfolio based on the given data. We first create a column to hold the portfolio weights of the previous day and create a column that can help us check the portfolio weights of the previous day holding. We merged the previous day's holdings and initialized them to zero weights when the backtest first starts. We then defined a new function called build_universe, which can filter the market cap of companies larger than one billion dollars no matter current or the past.

In order to find the optimal portfolio, we need to compare the outcome of different sets of factors. We defined the “setdiff” function which generates the non-alpha (risk) factor. And considering factor risk. We set risk aversion (k) as 1.0e-6, X is a matrix of risk factor exposure, and F is factor covariance matrix. To improve the computation efficiency, we used matrix factorization technique, wherein $Q := F^{1/2} X^T$ is defined so that $Q^T Q = X F X^T$. We considered idiosyncratic risk as the next step. We proceeded with a diagonalized version of the squared volatility of stocks (factor covariance) matrix as D. We implement a function called diagonal_factor_cov to perform the matrix creating process. Then, We multiplied 0.01 to SpecRisk since specific risk data was in percent in the raw data to get the specific variance for each stock in the universe.

```
#P5
specVar = (0.01 * universe['SpecRisk']) ** 2
```

After that, we calculated the expected portfolio return. We used an equal-weighted average of chosen alphas to combine alphas. The combines alpha vector is supposed to be indicative of the component of future asset returns that is not explained by common factors. For the last term, transaction costs, we brought up the idea of average daily volume, since orders which are large in this metric are predicted to cause a greater impact on the price. We have to multiply the price change due to market impact by the number of dollars traded in order to calculate the transaction cost.

Thus, the objective function is given as factor risk + idiosyncratic risk - expected portfolio return + transaction costs. We want to maximize the objective, or equivalently minimize the negative of the objective.

$$f(h) = \frac{1}{2}\kappa h^T Q^T Q h + \frac{1}{2}\kappa h^T D h - \alpha^T h + (h - h_0)^T \Lambda (h - h_0)$$

We used a popular machine learning optimization algorithm called Limited-memory BFGS (L-BFGS)  for parameter estimation. L-BFGS is an optimizer from the quasi-Newton methods family, which approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) using a limited amount of computer memory. This process is recommended because it could improve performance when the gradient is calculated and be taken by the optimizer.

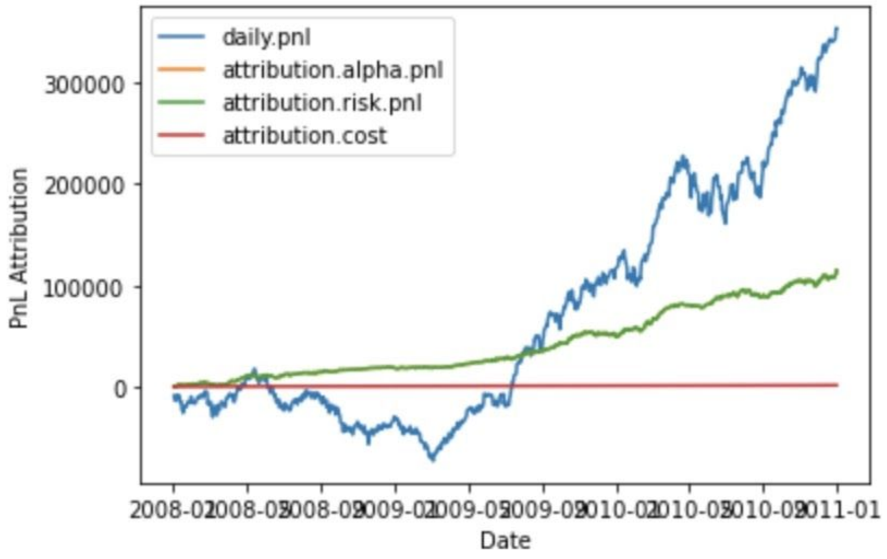
At last, we defined the function “form_optimal_portfolio”. It combined all functions that were defined above, that could optimize our portfolio. The function will return the optimized portfolio, the risk exposures, the alpha exposures and the total cost.

## Backtest

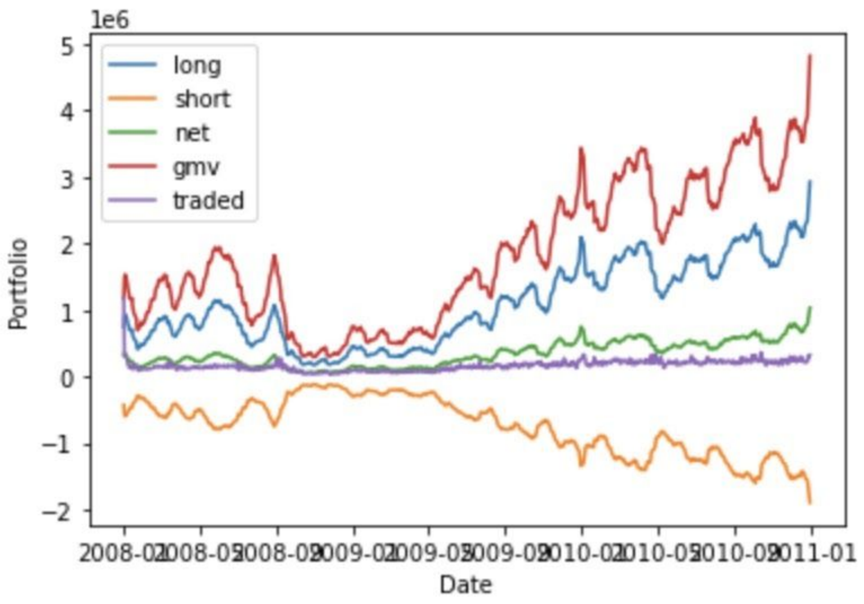
The next step is to build the trade list. The trade list is the difference between the most recent optimal asset holdings and the previous day's optimal holdings. The function takes the previous holdings, and the result of our optimization function, and outputs a trade list.

We then calculated the optimal portfolio holdings and trade list. First, we import the tqdm package. The package makes creating progress bars very simple as well as shows up in Jupyter notebooks. We define a function that takes the current optimal holdings and returns them as the previous optimal holdings. Next, we ran the backtest through each day by calculating the optimal portfolio holdings and trade list.

After that, we created a function that plots the cumulative sum of the idiosyncratic contribution and risk factor contributions over time. P&L is the total of realized daily returns of the assets, weighted by the optimal portfolio holdings chosen, and added together to get the portfolio's profit and loss. To calculate the P&L attributed to the alpha factors, the risk factors, and the cost, we defined a function build_pnl_attribution. P&L attributed to the alpha factors is calculated as the factor returns times factor exposures for the alpha factors, and the P&L attributed to the risk factors is calculated as the factor returns times factor exposures of the risk factors. The graph shows the daily P&L and each attribution.



In the end, we calculated the total of long positions, short positions, net positions, gross market value, and amount of dollars traded. We defined a function called build_portfolio_characteristics to make the calculations. The graph shows the individual value trend of the mentioned characteristics throughout the period. The strategy is approximately equal amounts long and short, and net exposure is very low, which indicates self-financing.


1e6
long
short
net
gmv
traded
Portfolio
Date