COMS W4995-Topics Final Project

Ganghua Mei (UNI: gm3044)

July 4, 2022

I. Scope and Purpose

A. Context

The final project is written as part of the evaluation of the Machine Learning with Applications in Finance summer course at Columbia University. The portfolio manager is Ganghua Mei, with an initial $1 million AUM. The investment portfolio is simulated from June 13, 2022 to July 1, 2022, with a total of 15 trading days.

B. Investors

The Investment Policy Statement governs the personal investment portfolios of Ganghua Mei, who is also the portfolio manager.

II. Investment, Return, and Risk Objectives

A. Overall investment objective

The investment program governed by the IPS is intended to generate high returns to compete with other portfolio managers in class while acting highly disciplinary on risk management. However, the investment principles and associated philosophy may also shed light on individual investors aged between 25 and 50 with stable future cash flows that want to grow their investment portfolios fiercely and are willing to take a decent amount of risks. That is, a portfolio that generates high returns is functionally different from a retirement saving plan.

B. State return, distribution, and risk requirements

The final project and its IPS developed by Ganghua Mei is aims generate an average of 20 percent return per year irrespective of the market conditions.

1. To achieve this goal, the manager will only invest in the US equity market. In addition, given a bear market outlook in a 12-month horizon, the manager will exclusively select companies with a market cap of $8 billion or more to ensure these companies have high trading volume and receive enough attention from institutional and retail investors especially during market turmoils.

2. Based on the overall expected portfolio annual return of 20 percent, fees of 2 percent, inflation of 8 percent, and an effective tax rate of 37 percent of total appreciation, this portfolio may support an annual real spending rate of 3.34 percent of the portfolio market value while retaining the potential for capital preservation or nominal growth at a rate of 11.34 percent.

3. See the table below for the asset allocation plan for our portfolio. Specifically,

    a. Cash position will be no less then 20 percent of total allocation. Equity allocation will be ranged between 60 to 80 percent. No investment will be made to fixed income assets.

    b. Given the bear market outlook in a 12-month horizon, daily investments will not be greater than 10 percent of total asset to maintain buying power and take advantage of downward market movements.

    c. Re-balancing the portfolio every one-fifth of the investment periods (i.e., every two to three days for a total investment of 15 trading days) and re-evaluating the portfolio while taking the profits after gaining 6 percent of return for the total asset. This is to ensure the portfolio's positions can be built progressively while taking the gains cautiously during market turmoils.

d. Last but not least, the manager enforces a 9 percent stop-loss red line for each position he invests, meaning he will face a total loss of 2.25 percent (9 percent $\times$ 1/4) of the total asset given a portfolio consisting of 4 stocks. In other words, the manager will be broke even if another position gains a 9.8 percent return after triggering a stop loss. The likelihood is not low given the market volatility and during bear market rallies.

e. Margin usage and short sell are strictly prohibited in this portfolio to ensure no single position will trigger a massive margin call. This portfolio is intended to sustain liquidity for both manager and investor throughout the investment periods.

Asset Class	Subasset Class	Target Allocation
Equity		60 - 80%
	US	60 - 80%
	Non - US	
Fixed income		0%
	Investment grade	
	Below-investment grade	
Cash		20 - 40%

These five rules govern the portfolio allocation and re-balance policies.

C. The risk tolerance of the investor

The portfolio only consists of equity and cash. Therefore, the associated risks are higher than most other portfolios managed by commercial banks that hold a significant proportion of their fixed-income assets. While this is the case, the portfolio manager only invests in mid-large cap companies that are less volatile than small-cap companies to provide a decent risk-adjusted return as opposed to all other counterparts. The investor also acknowledges that an absolute loss of 33 percent of total asset is intolerable. Thus, the portfolio manager will notify the investor if absolute loss approaches to 28 percent of total assets and react accordingly.

D. Relevant constraints

The portfolio manager will provide a quarterly report summarizing the portfolio's performance to the investor. In addition, performance evaluation will be accessed every one-fifth of the investment periods by the portfolio manager, and compare to SP500 — the selected benchmark agreed mutually the portfolio manager and investor. The investor also need to inform the manager in advance to make deductions to their investment account.

E. Other relevant considerations

The general investment philosophy for this portfolio is two-fold. On the one hand, the portfolio manager wants to take full advantage of under-price/over-sold mid-large cap companies based on fundamental and technical analysis with the support of cutting-edge machine learning stock selection techniques. On the other hand, the portfolio manager actively applies professional investment principles emphasizing profit-taking and risk management. A combination of the two approaches would be more likely than not, probabilistically speaking, to deliver a tremendous risk-adjust return to the investor.

III. Fundamental analysis and machine learning approaches for portfolio optimization

The portfolio manager developed a three-stage process for portfolio construction:

A. The first stage is to select mid-large cap companies based on fundamental and technical analysis following Steven Downey (2020).

B. The second stage use neural networks to predict the stock returns in the spirit of Gu, Kelly and Xiu (2020)and Van Binsbergen, Han and Lopez-Lira (2020).

C. The third stage use predicted returns for mean-variance optimization and select stocks that maximize the sharpe ratio.

A. Fundamental and technical analysis

The portfolio manager first merged three pieces of datasets from Sharadar over the past five years, including Core US Fundamentals Data, Equity Prices Data, and Tickers Name.

Specifically, Sharadar's fundamentals data allows the manager to construct fundamental factors, including earnings yield, EBITDA/EV multiple, P/FCF Ratio, FCF/Assets ratios, shareholder yield, FCF/Assets, ROA, ROIC, gross margin, current ratio, and the EBITDA-to-interest coverage ratio.[1] These factors can provide valuable information revealing a company's short- and long-term financial conditions.

On the other hand, Sharadar's equity prices data allows the manager to quantify the volatility, trend (e.g., 200-day moving average), and recent momentum of these companies. The portfolio manager then constructed these measures in the datasets and aggregated them to a total score reflecting each company's overall quality and market performance in a 12-month period.

Figure 1 (top) presents the aggregated scores of 473 US companies with a market cap of at least $8 billion since 2018. Figure 1 (bottom) shows the manager's four positions for the first half of portfolio investment. The two additional selected criteria are that stock prices should be around their 52-week low in May 2022 to increase the chances of getting a short-term rebound, t and choose one position in each sector for diversification.

[1]See Appendix A.1 for a sample of datasets.

Figure 1: US Mid-large Cap Companies 2018-2022

	ticker	sector	Total Score
166	BIIB	Healthcare	15.247131
1376	LPX	Basic Materials	14.702598
692	MU	Technology	14.396380
345	EBAY	Technology	13.901160
917	SWKS	Technology	13.690754
...	...	...	...
1393	NTRA	Healthcare	-12.783486
1403	NVAX	Healthcare	-13.100797
1378	RUN	Technology	-15.629291
1315	LYFT	Technology	-18.375465
1416	GME	Consumer Cyclical	-213.490637

473 rows × 3 columns

	ticker	sector	Total Score
345	EBAY	Technology	13.901160
662	MNST	Consumer Defensive	8.657840
592	KNX	Industrials	5.652692
1330	ZM	Communication Services	4.004370

In the first 9 trading days, the portfolio manager made investment decisions based on stage one analysis only by constructing an equal-weighted portfolio in the spirit of Mei and McNown (2019).

B. Neural networks stock predictions

The portfolio manager is deeply inspired and motivated after taking W4995 summer course and reading through various articles, such as Ghayur, Heaney and Platt (2018), to implement machine learning algorithms for stock predictions.

FIGURE 2: Distribution of Top forty Companies by Sector

	sector
Technology	20
Healthcare	11
Consumer Cyclical	5
Basic Materials	2
Industrials	2
Energy	1
Consumer Defensive	1

The portfolio manager follows the stage one analysis to pick forty companies with the highest total scores in the calendar year 2021. Consequently, these forty companies have good short and long-term financial conditions, are less risky and have upward trends and good momentum factors. This approach is similar but different from recently published papers that use machine learning techniques to select fundamental and technical factors that provide the best prediction from stock prices (Gu, Kelly and Xiu, 2020),

Figure 2 presents the distribution of these fourth companies by sector. It shows that technology, healthcare, and consumer cyclical are the top three sectors with the most qualified companies on the list.

In the second stage, the portfolio manager wants to let the machine learning techniques decide which companies to invest in. So first, the portfolio managers merged Sharadar's fundamentals data with equity prices and converted the stock prices to returns from Apr 4, 2018 to June, 14, 2022. Figure 3 presents the table.

Figure 3: Stock Returns of the Top forty Companies

```
1 Best_returns_daily.sample(5)
```

ticker date	AMGN	ANET	BIIB	BIO	BKNG	DKS	EBAY	EXEL	FFIV	HOLX	HPQ	
2019-07-09	-0.003567	0.009522	0.014642	-0.013773	0.001125	-0.024384	0.010116	0.026406	0.004718	0.009928	-0.005304	-0.0
2020-10-23	-0.003597	0.000673	-0.006747	0.004611	0.017595	0.000840	0.028686	0.021072	0.008089	-0.000883	-0.011886	0.0
2019-10-21	0.001036	0.018410	0.015678	0.006671	0.004326	0.010302	0.011334	0.024361	0.001089	-0.014922	0.008902	-0.0
2021-01-14	0.016489	0.011213	0.051074	-0.018500	0.000078	0.019298	-0.007898	-0.024210	0.008972	-0.017598	-0.003501	0.0
2021-07-16	0.005393	-0.012509	-0.010787	0.012013	-0.011372	-0.037509	0.000587	-0.035037	0.003405	0.017113	-0.020641	-0.0

```
1 Best_returns_daily.columns
```

```
Index(['AMGN', 'ANET', 'BIIB', 'BIO', 'BKNG', 'DKS', 'EBAY', 'EXEL', 'FFIV',
       'HOLX', 'HPQ', 'INTU', 'LOGI', 'LPX', 'LRCX', 'META', 'MU', 'NTAP',
       'NVDA', 'REGN', 'RHI', 'STLD', 'SWKS', 'TPL', 'TXN', 'VRTX', 'WBD',
       'WSM'],
      dtype='object', name='ticker')
```

Second, the portfolio manager merged it with Fama-fench five factors data (i.e., Mkt-RF, SMB, HML, RMW, CMA), which have empirically shown to explain a significant proportion of stock returns. Figure 4 presents the table. The portfolio manager then created a lag value for each index return and used it to predict the current stock prices, similar to assignment 5.

Figure 4: Stock Returns with Fama-fench Five Factors

Date	Mkt-RF	SMB	HML	RMW	CMA	RF	AMGN	ANET	BIIB	BIO	...	NVDA	REGN	RHI	STLD	SWKS
2018-04-03	0.0124	-0.0003	0.0018	0.0026	0.0017	0.00007	0.013068	0.000000	-0.001536	0.006993	...	0.019434	0.004961	0.025052	0.028478	0.011395
2018-04-04	0.0117	0.0034	-0.0031	0.0006	-0.0012	0.00007	0.031506	0.021506	0.022216	0.005660	...	0.003958	0.015503	-0.006280	0.008625	0.020486
2018-04-05	0.0075	0.0005	0.0047	0.0010	0.0027	0.00007	-0.008760	0.014597	-0.027239	-0.012175	...	-0.021482	-0.025967	0.011614	0.024752	-0.008331
2018-04-06	-0.0219	0.0037	-0.0006	0.0014	-0.0004	0.00007	-0.022442	-0.015300	-0.027662	-0.013618	...	-0.032216	-0.027329	-0.035117	-0.045235	-0.032794
2018-04-09	0.0030	-0.0024	-0.0051	-0.0075	-0.0027	0.00007	0.008802	0.003015	-0.000505	0.002868	...	0.005414	0.000282	0.014348	0.004600	0.004814
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2022-05-24	-0.0123	-0.0046	0.0184	0.0141	0.0115	0.00001	0.011444	-0.025513	0.015747	-0.003879	...	-0.044029	0.006027	-0.026691	-0.000384	-0.023370
2022-05-25	0.0122	0.0074	0.0021	-0.0053	0.0000	0.00001	0.004446	0.012639	-0.000889	-0.019586	...	0.050823	0.007084	0.015222	0.032522	0.018222
2022-05-26	0.0218	0.0002	-0.0063	-0.0014	-0.0031	0.00001	0.000158	0.027340	0.009192	0.027312	...	0.051605	-0.010769	-0.005265	0.046503	0.027925
2022-05-27	0.0258	0.0002	-0.0130	-0.0063	-0.0025	0.00001	0.008733	0.020827	0.009695	0.047449	...	0.053778	0.013695	0.028305	0.040289	0.039889
2022-05-31	-0.0071	-0.0039	0.0044	0.0079	-0.0024	0.00001	0.005798	-0.033910	-0.030068	-0.017286	...	-0.007389	-0.040502	0.008728	-0.027452	0.001472

1049 rows × 34 columns

Third, the portfolio manager then performed a rolling window between Apr 4, 2018, and Apr 5,2021 to train the neural networks model and used the tuned model to train and predict the stock prices between Apr 6, 2021 and May 31, 2022.[2] The top nine stocks will be selected for mean-variance optimization. Figure 5 presents the table.

FIGURE 5: The Top nine stocks prior to Portfolio Diversification

	Avg Daily Return	Volatility	Sharpe Ratio	MSE
WSM_pred	0.066670	0.009603	6.942269	0.006151
BIIB_pred	0.053843	0.007978	6.748730	0.003669
TPL_pred	0.050324	0.013972	3.601862	0.003433
LRCX_pred	0.034508	0.010050	3.433561	0.002310
RHI_pred	0.033408	0.001680	19.886826	0.001649
HPQ_pred	0.033382	0.006759	4.939216	0.001807
BIO_pred	0.025382	0.001500	16.919366	0.001336
NTAP_pred	0.024630	0.002183	11.285036	0.001112
MU_pred	0.022125	0.000398	55.562861	0.001473

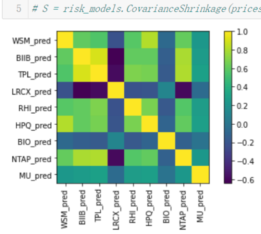
C. Mean-variance optimization

In this stage, the portfolio manager used stock predictions from stage two to calculate returns and the Ledoit-Wolf covariance matrix and maximize the sharpe ratio to obtain the portfolio weights for each position.

FIGURE 6: The Top Nine Stocks prior to Portfolio Diversification

```
mu = df_8['Avg Daily Return']
S = risk_models.CovarianceShrinkage(cov,returns_data=True,  frequency=252, log_returns=False).ledoit_wolf()
plotting.plot_covariance(S, plot_correlation=True);

# S = risk_models.CovarianceShrinkage(prices).ledoit_wolf()
```




```
OrderedDict([('WSM_pred', 0.00563),
             ('BIIB_pred', 0.07914),
             ('TPL_pred', 0.0),
             ('LRCX_pred', 0.07737),
             ('RHI_pred', 0.48174),
             ('HPQ_pred', 0.0),
             ('BIO_pred', 0.35611),
             ('NTAP_pred', 0.0),
             ('MU_pred', 0.0)])
```

[2]That is, he used the first 60 days as a training sample and the next 30 days as testing and then calculated the MSEs. The process is repeated from Apr 4, 2018 to Apr 5, 2021.

IV. The Evolution of the Portfolio

As the portfolio manager discussed in the previous section, he equally invested four stocks from different sectors in the first 9 trading days while re-balancing the portfolio on day 4 and day 6.[3] Figure 7 presents the holding positions over the investment period. He then closed all his positions to take profits at the end of day 9 because the cumulative return was above 6 %.[4] He started to invest in 5 different new positions based on stage two's results. The associated portfolio weights are calculated using stage three's results and made two re-balancing on Day 12 and Day 14, respectively.

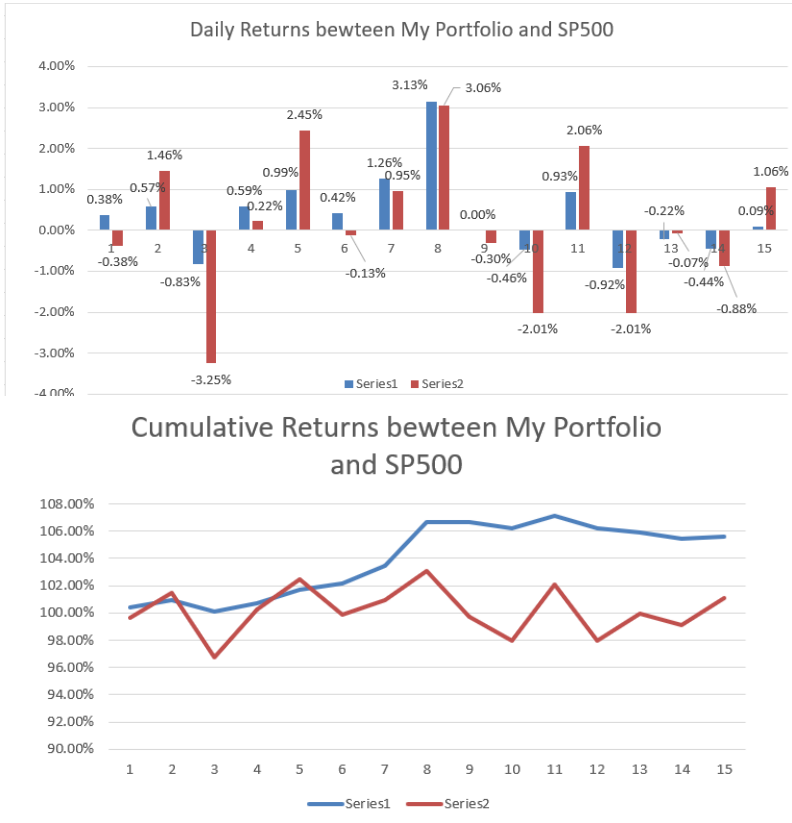
Figure 7: The Holding Position of Portfolio

Date	Holdings						Shares					
6/13/2022	Cash	EBAY	KNX	MNST	ZM			1200	1200	600	600	
6/14/2022	Cash	EBAY	KNX	MNST	ZM			1200	1200	600	600	
6/15/2022	Cash	EBAY	KNX	MNST	ZM			1200	1200	600	600	
6/16/2022	Cash	EBAY	KNX	MNST	ZM			2400	2400	1200	1200	
6/17/2022	Cash	EBAY	KNX	MNST	ZM			2400	2400	1200	1200	
6/21/2022	Cash	EBAY	KNX	MNST	ZM			3600	3600	2400	2400	
6/22/2022	Cash	EBAY	KNX	MNST	ZM			3600	3600	2400	2400	
6/23/2022	Cash	EBAY	KNX	MNST	ZM			3600	3600	2400	2400	
6/24/2022	Cash	EBAY	KNX	MNST	ZM			3600	3600	2400	2400	
6/24/2022	Cash	BIIB	LRCX	RHI	WSM	BIO		100	50	1000	150	70
6/27/2022	Cash	BIIB	LRCX	RHI	WSM	BIO		200	100	2000	300	140
6/27/2022	Cash	BIIB	LRCX	RHI	WSM	BIO		200	100	2000	300	140
6/28/2022	Cash	BIIB	LRCX	RHI	WSM	BIO		300	150	3000	400	180
6/29/2022	Cash	BIIB	LRCX	RHI	WSM	BIO		300	150	3000	400	180
6/30/2022	Cash	BIIB	LRCX	RHI	WSM	BIO		400	200	4000	500	220
7/1/2022	Cash	BIIB	LRCX	RHI	WSM	BIO		400	200	4000	500	220

Figure 8 presents our portfolio's daily and cumulative returns versus the benchmark SP500. Two features worth mentioning: 1. increasing the holding progressively will take advantage of market movements, particularly in the downturn from day 10 to day 15; 2. keeping the daily loss small and taking the profits regularly are keys to beating the market.

[3]See Rule b for in section II.B for more details.
[4]See Rule c for in section II.B for more details.

Figure 8: The Daily and Cumulative Returns of Portfolio Versus SP500



V. The Performance of the Portfolio

As you may see in Figure 9, the average daily return of our portfolio in the investment period is .367% (or 148.5% annually) versus 0.148% (or 39.905% annually). In particular, the total cumulative return is 5.57% which is 2.75 times the returns of SP500. Moreover, our portfolio is much less volatile (0.010) than the SP500 (0.018) with a sharpe ratio of .35.[5] I then computed an univariate CAPM model to calculate the associated alpha and beta. Alpha is 0.003 and beta is 0.488; both are statistically significant at the 10 percent level, meaning our portfolio has generated excess return over SP500 while the correlation between our portfolio and SP500 is relatively low.[6] Lastly, this portfolio's active

[5] The sharpe ratio is calculated as (average daily return - risk free rate/252)/total volatility.

[6] Excess returns for the portfolio and the market are calculated using converted daily average risk-free rate (annual average risk-free rate (10-year Treasury Rate)/252).

risk(standard deviation of the tracking error) is low (0.01 and with a decent information ratio (0.276). Overall, these numbers tell me that the portfolio constructed using the three-stage approach performs very well related to the SP500 in the investment period.

FIGURE 9: Descriptive Statistics

	Portfolio	Benchmark
Arithmetic Average	0.367%	0.148%
Geometric Average	0.362%	0.133%
Total Cumulative Return	5.570%	2.019%
Annualized Cumulative Return	148.582%	39.905%
Total Volatility	0.01008	0.01755
Annualized Volatility	0.16007	0.27865
Sharpe ratio	0.35224	0.07761
Alpha	0.00289	
Beta	0.48778	
Active Risk	0.01045	
Information Ratio	0.27629	
Correlation to Benchmark	0.84910	

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.849099977
R Square	0.720970771
Adjusted R Square	0.699506984
Standard Error	0.005527639
Observations	15

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	1	0.001026339	0.00102634	33.590101	6.21943E-05
Residual	13	0.000397212	3.0555E-05		
Total	14	0.001423551			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	0.002887402	0.001431828	2.01658499	0.0648822	-0.000205873	0.005980678	-0.000205873	0.005980678
X Variable 1	0.487778445	0.084162175	5.79569678	6.219E-05	0.305957121	0.66959977	0.305957121	0.66959977

Figure 10 presents the performance attribution of our portfolio. It turns out that stock and cash are roughly 60/40 split such that the total excess return of the portfolio is 1.31% (or 24.69% annually) after taking the performance of SP500 into account.[7]

[7]Annual excess return is calculated as $(1 + 1.31\%/15)^{252} - 1$.

FIGURE 10: Descriptive Statistics

Asset Allocation

	Portfolio weight	Benchmark weight	Excess weight	Benchmark return	Contribution
Stocks	59.84%	100%	-40.16%	2.02%	-0.81%
Cash	40.16%	0%	40.16%	0	0
Contribution of Asset Allocation					-0.81%

Stock selection

	Portfolio performance	Benchmark performance	Excess performance	Portfolio weight	Contribution
Contribution of stock selection	5.57%	2.02%	3.55%	59.84%	2.12%
Total excess return of portfolio					1.31%

Last but not least, the portfolio manager conducted a Fama french 5-factor model analysis using the previous day's portfolio weights to back-test portfolio returns between March 1,2022 and May 27, 2022. Figure 11 presents the data (top) and estimated results (bottom). Specifically, the portfolio manager failed to reject the portfolio's excess return as zero (p=0.599), which is consistent with the theory. Moreover, the correlation between the portfolio and the market is still relatively low (.65), and only SMB, RMW(Robust minus Weak), and CMA(Conservative minus Aggressive) are statistically significant at the 10 percent level. In particular, RMW is the only factor that is statistically significant at the 5 percent level. In other words, we found that companies with high operation profitability relative to those that do not provide excess returns. This is very sensible because institutional and retail investors want to invest in companies that will stay afloat and constantly generate profits, particularly during stock market turmoils (e.g., the first half of 2022).

FIGURE 11: Descriptive Statistics

```
1  FF_5.head(5)
```

Date	Mkt-RF	SMB	HML	RMW	CMA	RF	Port_Ret	RP-Rf
2022-03-01	-0.0156	-0.0028	-0.0083	-0.0037	0.0059	0.0	-0.019785	-0.019785
2022-03-02	0.0181	0.0062	0.0133	0.0047	0.0042	0.0	0.010955	0.010955
2022-03-03	-0.0088	-0.0021	0.0151	0.0124	0.0120	0.0	0.001506	0.001506
2022-03-04	-0.0110	-0.0049	0.0068	0.0041	0.0121	0.0	-0.009954	-0.009954
2022-03-07	-0.0312	0.0071	0.0090	-0.0069	0.0136	0.0	-0.031365	-0.031365

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  RP-Rf   R-squared:                       0.841
Model:                            OLS   Adj. R-squared:                  0.827
Method:                 Least Squares   F-statistic:                     60.18
Date:                Mon, 04 Jul 2022   Prob (F-statistic):           1.73e-21
Time:                        16:46:28   Log-Likelihood:                 241.12
No. Observations:                  63   AIC:                            -470.2
Df Residuals:                      57   BIC:                            -457.4
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0004      0.001     -0.529      0.599      -0.002       0.001
Mkt-RF         0.6520      0.053     12.402      0.000       0.547       0.757
SMB            0.2477      0.147      1.685      0.097      -0.047       0.542
HML            0.0213      0.111      0.192      0.849      -0.201       0.243
RMW            0.2640      0.106      2.501      0.015       0.053       0.475
CMA           -0.2886      0.161     -1.788      0.079      -0.612       0.035
==============================================================================
Omnibus:                       10.524   Durbin-Watson:                   2.222
Prob(Omnibus):                  0.005   Jarque-Bera (JB):               18.836
Skew:                          -0.463   Prob(JB):                     8.13e-05
Kurtosis:                       5.513   Cond. No.                         303.
==============================================================================
```

VI. Analysis and conclusions

A. The first 9 days of trading MNST and ZM surprised me the most. These two positions have an average cumulative return of 15 percent in 9 days. Since I considered an equal-weighted portfolio, half of the investment benefited from these two positions. The leading cause stems from the fact that both stocks dropped 30 percent in the first five months of 2022 and have been significantly oversold and underpriced, given the fundamentals calculated in stage one analysis.

B. On days 10 to 15, LRCX surprised me the most. This stock also dropped 30 percent in the first five months. However, it continued to drop another 12.2 percent in 5 trading days after I opened a position, which wiped out all the gains after day 10.

The leading cause is that the semiconductor industry revealed very weak guidance in the second half of 2022 following MU's recently Q2 earnings call.

C. At a high level, both fundamental and technical analysis are significant factors to investigate before making any investment, particularly during high market volatility periods. Moreover, machine learning algorithms building upon fundamental and technical analysis can provide excellent guidance for stock selection.

D. I will definitely incorporate the two approached I learned and developed from W4995 to my own investment. In fact, I already did.

References

Ghayur, Khalid, Ronan Heaney, and Stephen Platt. 2018. "Constructing long-only multifactor strategies: portfolio blending vs. signal blending." *Financial Analysts Journal*, 74(3): 70–85.

Gu, Shihao, Bryan Kelly, and Dacheng Xiu. 2020. "Empirical asset pricing via machine learning." *The Review of Financial Studies*, 33(5): 2223–2273.

Mei, Ganghua, and Robert McNown. 2019. "Dynamic causality between the US stock market, the Chinese stock market and the global gold market: Implications for individual investors' diversification strategies." *Applied Economics*, 51(43): 4742–4756.

Steven Downey, CFA. 2020. "How to build a multi-factor equity portfolio in Python."

Van Binsbergen, Jules H, Xiao Han, and Alejandro Lopez-Lira. 2020. "Man vs. machine learning: The term structure of earnings expectations and conditional biases." National Bureau of Economic Research.

Appendix Below

In [69]:
```python
import os
import datetime
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas_datareader.data as web

# Main reference of Fina Project_1 https://medium.com/swlh/how-to-build-a-multi-factor-equity-portfolio-in-pyth
```

In [70]:
```python
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 200)
plt.style.use('ggplot')
pd.options.mode.chained_assignment = None

fundamental_data = (pd.read_csv('SHARADAR_SF1_FUN.csv'))
equity_prices = (pd.read_csv('SHARADAR_SEP.csv'))
tickers_df = (pd.read_csv('SHARADAR_TICKERS.csv', low_memory=False))
```

In [71]:
```python
tickers_df1 = tickers_df[tickers_df['location'].notnull()]
tickers_df1 = tickers_df1[tickers_df1['location'].str.contains("U.S.")]
tickers_df1 = (tickers_df1[['ticker', 'sector', 'name',
               'industry', 'scalemarketcap']])
myset_ticker = set(tickers_df1.ticker)
list_tickers = list(myset_ticker)

USA_fundamentals = fundamental_data[fundamental_data['ticker'].isin(list_tickers)]

sector_stocks = tickers_df1[tickers_df1['sector'] != 'Real Estate']
sector_stocks = sector_stocks[sector_stocks['sector'] != 'Financial Services']
sector_tickers = sector_stocks['ticker'].tolist()

fundamentals = USA_fundamentals[USA_fundamentals.ticker.isin(sector_tickers)]
fundamentals = fundamentals[fundamentals.dimension == 'ART']

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


Data_for_Portfolio = fundamentals[fundamentals['marketcap'] >= 7e9]
tickers = Data_for_Portfolio['ticker'].tolist()
print('There are ' + str(len(set(tickers))) + ' tickers') #number of unique tickers

#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)
```

```
There are 1000 tickers
```

Portfolio Construction Starting Date

In [72]:
```python
Data_for_Portfolio= Data_for_Portfolio[Data_for_Portfolio['calendardate'] > '2017-12-31']
```

Constructing factors

In [73]:
```python
### 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'])

####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']

Data_for_Portfolio = Data_for_Portfolio.dropna()

Sector_stock_prices = equity_prices.loc  \
    [equity_prices['ticker'].isin(tickers)]
```

In [74]:
```python
f_date = datetime.date(2018, 3, 31)
l_date = datetime.date(2022, 6, 14) #choosing the last date, results in last
delta = l_date - f_date
quarters_delta = np.floor(delta.days/(365/4))
quarters_delta = int(quarters_delta)
first_quarter = str('2018-03-31') #using f_date
Data_for_Portfolio_master = pd.DataFrame(Data_for_Portfolio)
```

In [75]:
```python
Winsorize_Threshold = .025

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

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()
```

In [76]:
```python
Data_for_Portfolio_master_filter = pd.DataFrame()

for i in range(0, quarters_delta, 4):

    Date = pd.to_datetime(first_quarter) + pd.tseries.offsets.QuarterEnd(i)
    Date = Date.strftime('%Y-%m-%d')
    df=Data_for_Portfolio_master.loc[Data_for_Portfolio_master['calendardate'] == Date]
    Data_for_Portfolio_master_filter=Data_for_Portfolio_master_filter.append(df)
```

Creating all the factors

In [77]:
```python
###### VALUE FACTOR ######

#Winsorize the metric data and compress outliers if desired
Data_for_Portfolio_master_filter['E/P Winsorized'] = stats.mstats.winsorize(Data_for_Portfolio_master_filter['E
Data_for_Portfolio_master_filter['EBITDA/EV Winsorized'] = stats.mstats.winsorize(Data_for_Portfolio_master_fil
Data_for_Portfolio_master_filter['FCF/P Winsorized'] =  stats.mstats.winsorize(Data_for_Portfolio_master_filter

#create Z score to normalize the metrics
Data_for_Portfolio_master_filter['E/P Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['E/P Winsorized'])
Data_for_Portfolio_master_filter['EBITDA/EV Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['EBITDA/EV Winsorized'])
Data_for_Portfolio_master_filter['FCF/P Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['FCF/P Winsorized'])

Data_for_Portfolio_master_filter['Valuation Score'] = \
        Data_for_Portfolio_master_filter['E/P Z score'] \
        + Data_for_Portfolio_master_filter['EBITDA/EV Z score']\
        + Data_for_Portfolio_master_filter['FCF/P Z score']


###### QUALITY FACTOR ######

Data_for_Portfolio_master_filter['FCF/Assets Winsorized'] = \
    stats.mstats.winsorize(Data_for_Portfolio_master_filter['FCF/Assets'], \
                           limits=Winsorize_Threshold)
Data_for_Portfolio_master_filter['ROA Winsorized'] = \
    stats.mstats.winsorize(Data_for_Portfolio_master_filter['ROA'], \
                           limits=Winsorize_Threshold)
Data_for_Portfolio_master_filter['ROIC Winsorized'] = \
    stats.mstats.winsorize(Data_for_Portfolio_master_filter['ROIC'], \
                            limits=Winsorize_Threshold)
Data_for_Portfolio_master_filter['Gross Margin Winsorized'] = \
    stats.mstats.winsorize(Data_for_Portfolio_master_filter['GROSS MARGIN'], \
                           limits=Winsorize_Threshold)
Data_for_Portfolio_master_filter['Current Ratio Winsorized'] = \
    stats.mstats.winsorize(Data_for_Portfolio_master_filter['CURRENT RATIO'], \
                           limits=Winsorize_Threshold)
Data_for_Portfolio_master_filter['Interest/EBITDA Winsorized'] = \
    stats.mstats.winsorize(Data_for_Portfolio_master_filter['INTEREST/EBITDA'], \
                           limits=Winsorize_Threshold)

#create Z score

Data_for_Portfolio_master_filter['FCF/Assets Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['FCF/Assets Winsorized'])
Data_for_Portfolio_master_filter['ROA Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['ROA Winsorized'])
Data_for_Portfolio_master_filter['ROIC Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['ROIC Winsorized'])
Data_for_Portfolio_master_filter['Gross Margin Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['Gross Margin Winsorized'])
Data_for_Portfolio_master_filter['Current Ratio Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['Current Ratio Winsorized'])
Data_for_Portfolio_master_filter['Interest/EBITDA Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['Interest/EBITDA Winsorized'])

Data_for_Portfolio_master_filter['Quality Score'] = \
    Data_for_Portfolio_master_filter['FCF/Assets Z score'] \
        + Data_for_Portfolio_master_filter['ROA Z score'] \
        + Data_for_Portfolio_master_filter['ROIC Z score']\
        + Data_for_Portfolio_master_filter['Gross Margin Z score']\
        + Data_for_Portfolio_master_filter['Current Ratio Z score']\
        - Data_for_Portfolio_master_filter['Interest/EBITDA Z score']

###### SHAREHOLDER YIELD FACTOR #####

Data_for_Portfolio_master_filter['Shareholder Yield Winsorized'] = \
    stats.mstats.winsorize(Data_for_Portfolio_master_filter['Shareholder Yield'], \
                           limits=Winsorize_Threshold)
Data_for_Portfolio_master_filter['Shareholder Yield Z score'] = \
    stats.zscore(Data_for_Portfolio_master_filter['Shareholder Yield Winsorized'])
Data_for_Portfolio_master_filter['Shareholder Yield Score'] = \
    Data_for_Portfolio_master_filter['Shareholder Yield Z score']

###### LOW VOLATILITY FACTOR ######

#must have fundamental data from previous factors for price based factors
#as some equities have price data and no fundamental data which should not
#be included
Sector_stocks_Fundamental_tickers = Data_for_Portfolio_master_filter['ticker'].tolist()

Sector_stock_prices_vol_df = Sector_stock_prices.loc\
    [Sector_stock_prices['ticker'].isin(Sector_stocks_Fundamental_tickers)]

Sector_stock_prices_vol_df_1 = Sector_stock_prices_vol_df.iloc[:, [0, 1, 5]]

Sector_stock_prices_vol_df_1_wide = Sector_stock_prices_vol_df_1.pivot\
    (index='date', columns='ticker', values='close')

Sector_stock_prices_vol_df_1_wide = Sector_stock_prices_vol_df_1_wide.fillna(0)
Sector_stock_returns =  Sector_stock_prices_vol_df_1_wide.pct_change()

#create rolling vol metric for previous 2 years
Sector_stock_rolling_vol = Sector_stock_returns.rolling(252*2).std()

#Choose second to last trading day to look at previous vol
#Sometimes the dates are off when trying to line up end of quarter and business
#days so to eliminate errors in the for loop I go to day of quarter, shift forward
#a business day and then go back two business days
Date_to_execute_trade = pd.to_datetime(Date) + pd.tseries.offsets.QuarterEnd()
Date_to_execute_trade_plus1 = Date_to_execute_trade + pd.tseries.offsets.BusinessDay(1)
final_trade_date = Date_to_execute_trade_plus1 - pd.tseries.offsets.BusinessDay(2)

#pick the final trade date volatility for each ticker
Filter_Date_Vol = final_trade_date.strftime('%Y-%m-%d')
Filter_Vol_Signal = Sector_stock_rolling_vol.loc[Filter_Date_Vol]
Filter_Vol_Signal_Sort = Filter_Vol_Signal.sort_values().dropna()

#create z score and rank for the Volatility Factor
frame = { 'Vol': Filter_Vol_Signal_Sort}
Filter_Vol_Signal_df = pd.DataFrame(frame)
Filter_Vol_Signal_df['Vol Z Score'] = stats.zscore(Filter_Vol_Signal_Sort)
Filter_Vol_Signal_df = Filter_Vol_Signal_df.reset_index()

Data_for_Portfolio_master_filter = Data_for_Portfolio_master_filter.merge(Filter_Vol_Signal_df, how = 'inner',

###### TREND FACTOR #####

tickers_trend = list(Sector_stock_prices_vol_df_1_wide.columns)

#This is a very simply way to see how much a stock is in a trend up or down
#You could easily make this more complex/robust but it would cost you in
#execution time
df_sma_50 = Sector_stock_prices_vol_df_1_wide.rolling(50).mean()
df_sma_100 = Sector_stock_prices_vol_df_1_wide.rolling(100).mean()
df_sma_150 = Sector_stock_prices_vol_df_1_wide.rolling(150).mean()
df_sma_200 = Sector_stock_prices_vol_df_1_wide.rolling(200).mean()

#Get the same date for vol measurement near rebalance date
Filter_Date_Trend = final_trade_date.strftime('%Y-%m-%d')
Filter_Trend_Signal_50 = df_sma_50.loc[Filter_Date_Trend]
Filter_Trend_Signal_100 = df_sma_100.loc[Filter_Date_Trend]
Filter_Trend_Signal_150 = df_sma_150.loc[Filter_Date_Trend]
Filter_Trend_Signal_200 = df_sma_200.loc[Filter_Date_Trend]

Price_Signal = Sector_stock_prices_vol_df_1_wide.loc[Filter_Date_Trend]

Filter_SMA_Signal_df = pd.DataFrame(tickers_trend)
Filter_SMA_Signal_df = Filter_SMA_Signal_df.rename(columns={0: "ticker"})
Filter_SMA_Signal_df['SMA 50 position'] = np.where(Price_Signal > Filter_Trend_Signal_50,1,0)
Filter_SMA_Signal_df['SMA 100 position'] = np.where(Price_Signal > Filter_Trend_Signal_100,1,0)
Filter_SMA_Signal_df['SMA 150 position'] = np.where(Price_Signal > Filter_Trend_Signal_150,1,0)
Filter_SMA_Signal_df['SMA 200 position'] = np.where(Price_Signal > Filter_Trend_Signal_200,1,0)
Filter_SMA_Signal_df['Trend Score'] = np.mean(Filter_SMA_Signal_df, axis=1)
Data_for_Portfolio_master_filter = Data_for_Portfolio_master_filter.merge(Filter_SMA_Signal_df[['ticker','Trend

###### MOMENTUM FACTOR #####

tickers_momentum = list(Sector_stock_prices_vol_df_1_wide.columns)
#from the academic literature of 12 months - 1 month momentum
df_mom_11_months = Sector_stock_prices_vol_df_1_wide.pct_change(22*11)

Filter_Date_Mom = Date_to_execute_trade_plus1 - pd.tseries.offsets.BusinessDay(24)
Filter_Date_Mom_trim = final_trade_date.strftime('%Y-%m-%d')
Filter_Mom_Signal = df_mom_11_months.loc[Filter_Date_Mom_trim]

Filter_MOM_df = pd.DataFrame(tickers_momentum)
Filter_MOM_df = Filter_MOM_df.rename(columns={0: "ticker"})
Filter_MOM_df['Percent Change'] = Filter_Mom_Signal.values

Filter_MOM_df = Filter_MOM_df.replace([np.inf, -np.inf], np.nan)
Filter_MOM_df = Filter_MOM_df.dropna()
Filter_MOM_df['Momentum Score'] = stats.zscore(Filter_MOM_df['Percent Change'])

Data_for_Portfolio_master_filter = Data_for_Portfolio_master_filter.merge(Filter_MOM_df[['ticker','Momentum Sco
```

```
C:\Users\DELL\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3472: FutureWarning: Dropping of nuisance c
olumns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

In [80]:
```python
Data_for_Portfolio_master_filter['Total Score'] = \
    Data_for_Portfolio_master_filter['Valuation Score'] +  \
    Data_for_Portfolio_master_filter['Quality Score'] + \
    Data_for_Portfolio_master_filter['Shareholder Yield Score'] -  \
    Data_for_Portfolio_master_filter['Vol Z Score'] * \
    (Data_for_Portfolio_master_filter['Momentum Score'] + \
    Data_for_Portfolio_master_filter['Trend Score'])

number_firms = Data_for_Portfolio_master_filter.shape
number_firms = number_firms[0]
```

In [81]:
```python
Data_for_Portfolio_master_filter = \
    Data_for_Portfolio_master_filter.sort_values('Total Score', ascending=False)
Test=Data_for_Portfolio_master_filter[Data_for_Portfolio_master_filter['Total Score']>=8]
```

In [143…
```python
Data_for_Portfolio_master_filter[['ticker','sector','Total Score']].drop_duplicates(['ticker'],keep = 'first')
```

Out[143…

	ticker	sector	Total Score
166	BIIB	Healthcare	15.247131
1376	LPX	Basic Materials	14.702598
692	MU	Technology	14.396380
345	EBAY	Technology	13.901160
917	SWKS	Technology	13.690754
...	...	...	...
1393	NTRA	Healthcare	-12.783486
1403	NVAX	Healthcare	-13.100797
1378	RUN	Technology	-15.629291
1315	LYFT	Technology	-18.375465
1416	GME	Consumer Cyclical	-213.490637

473 rows × 3 columns

In [139…
```python
my_stocks = ['EBAY','KNX','MNST','ZM']
Data_for_Portfolio_master_filter[['ticker','sector','Total Score']][Data_for_Portfolio_master_filter['ticker'].
```

Out[139…

	ticker	sector	Total Score
345	EBAY	Technology	13.901160
662	MNST	Consumer Defensive	8.657840
592	KNX	Industrials	5.652692
1330	ZM	Communication Services	4.004370

In [ ]:
```python
# Data_for_Portfolio_master_filter[['ticker','sector','Total Score']][Data_for_Portfolio_master_filter['calenda
```

In [146…
```python
Test=Test.drop_duplicates(['ticker'])
Test['sector'].value_counts().to_frame().style.bar()
```

Out[146…

	sector
Technology	20
Healthcare	11
Consumer Cyclical	5
Basic Materials	2
Industrials	2
Energy	1
Consumer Defensive	1

In [148…
```python
First =  Data_for_Portfolio_master_filter[:40].drop_duplicates(['ticker'])
Last = Data_for_Portfolio_master_filter[-40:].drop_duplicates(['ticker'])
```

In [149…
```python
First_tickers = First['ticker'].tolist()
Last_tickers = Last['ticker'].tolist()
```

Merge selected stocks with equity price data

In [150…
```python
Best = Sector_stock_prices.loc\
    [Sector_stock_prices['ticker'].isin(First_tickers)]
Best = Best.iloc[:, [0, 1, 5]]
Worst = Sector_stock_prices.loc\
    [Sector_stock_prices['ticker'].isin(Last_tickers)]
Worst = Worst.iloc[:, [0, 1, 5]]
```

In [151…
```python
Best.sort_values(by='date', ascending=True)
```

Out[151…

	ticker	date	close
13885369	VRTX	2012-01-03	32.23
3862567	LRCX	2012-01-03	36.75
14007894	RHI	2012-01-03	28.66
16261629	STLD	2012-01-03	14.03
13885524	WSM	2012-01-03	36.94
...	...	...	...
2331919	HOLX	2022-06-14	67.87
2308612	EBAY	2022-06-14	42.58
2109248	BIIB	2022-06-14	194.18
6816252	MU	2022-06-14	58.70
6880335	TPL	2022-06-14	1636.32

77768 rows × 3 columns

In [152…
```python
#add a quarter to reporting so no lookahead bias
Date_to_execute_trade = pd.to_datetime(Date) + pd.tseries.offsets.QuarterEnd()
Date_to_execute_trade_plus1 = Date_to_execute_trade + pd.tseries.offsets.BusinessDay(1)
final_trade_date = Date_to_execute_trade_plus1 - pd.tseries.offsets.BusinessDay(1)

#add 4 quarters to end so the rebalance will be annual
end_date = Date_to_execute_trade + pd.tseries.offsets.QuarterEnd(4)
final_trade_date_trim = final_trade_date.strftime('%Y-%m-%d')
end_date_trim = end_date.strftime('%Y-%m-%d')
start_date = final_trade_date_trim
end_date = end_date_trim


#make data from long format to wide and fill in Na's with 0
Best_wide = Best.pivot\
    (index='date', columns='ticker', values='close')
Best_wide = \
    Best_wide.fillna(0)
Worst_wide = \
    Worst.pivot(index='date', columns='ticker', values='close')
Worst_wide = \
    Worst_wide.fillna(0)
```

In [153…
```python
if Best_wide.empty == True:
    days = price_index
    number_days = len(price_index)
    filler_returns = np.repeat(0, number_days)
    df = pd.DataFrame({'Days': days, 'Returns': filler_returns})
    df = df.set_index('Days')
    Best_wide = df
else:
    pass

if Worst_wide.empty == True:
    days = price_index
    number_days = len(price_index)
    filler_returns = np.repeat(0, number_days)
    df = pd.DataFrame({'Days': days, 'Returns': filler_returns})
    df = df.set_index('Days')
    Worst_wide = df
else:
    pass
```

In [154…
```python
start_date = str('2018-03-31')
start_date,end_date
```

Out[154…
```
('2018-03-31', '2022-06-30')
```

In [155…
```python
#pick out start date and end date for calculating equity returns
Best_wide.loc[start_date:end_date].head()
Best_wide = \
    Best_wide.loc[start_date:end_date]
Best_returns_daily = Best_wide.pct_change()


#if there are no assets then the Cheap_returns_daily become NaN and in that
#case we will need to by pass the normal operations and just keep the dataframe
x = np.logical_and\
    (Best_returns_daily.shape[1] == 1, Best_returns_daily.isnull().all() == True)

if x[0] == True:
    Best_returns_daily = Best_wide
else:
    #get rid of first NaN row
    Best_returns_daily = Best_returns_daily.dropna(how='all')

    #get rid of stocks that have no trading
    Best_returns_daily = Best_returns_daily.dropna(axis='columns')

column_length = Best_returns_daily.shape[1]
```

In [156…
```python
#pick out start date and end date for calculating equity returns
Worst_wide.loc[start_date:end_date].head()
Worst_wide = \
    Worst_wide.loc[start_date:end_date]
Worst_returns_daily = Worst_wide.pct_change()


#if there are no assets then the Cheap_returns_daily become NaN and in that
#case we will need to by pass the normal operations and just keep the dataframe
x = np.logical_and\
    (Worst_returns_daily.shape[1] == 1, Worst_returns_daily.isnull().all() == True)

if x[0] == True:
    Worst_returns_daily = Worst_wide
else:
    #get rid of first NaN row
    Worst_returns_daily = Worst_returns_daily.dropna(how='all')

    #get rid of stocks that have no trading
    Worst_returns_daily = Worst_returns_daily.dropna(axis='columns')

column_length = Worst_returns_daily.shape[1]
```

In [157…
```python
Best_returns_daily.to_csv('B.csv')
Worst_returns_daily.to_csv('W.csv')
```

In [165…
```python
Best_returns_daily.head(5)
```

Out[165…

ticker	AMGN	ANET	BIIB	BIO	BKNG	DKS	EBAY	EXEL	FFIV	HOLX	HPQ	INTU	LO
date													
2018-04-03	0.013068	0.000000	-0.001536	0.006993	0.018076	0.011670	0.006352	-0.009772	0.011584	0.003821	0.008515	0.003473	0.00863
2018-04-04	0.031506	0.021506	0.022216	0.005660	0.001664	0.039337	0.004544	0.048402	0.007517	0.013322	0.010319	0.015135	-0.00358
2018-04-05	-0.008760	0.014597	-0.027239	-0.012175	0.009696	0.012237	0.006032	-0.009413	-0.001185	0.004830	0.000464	-0.007050	0.00304
2018-04-06	-0.022442	-0.015300	-0.027662	-0.013618	-0.027584	-0.050886	-0.023482	-0.045701	-0.017663	-0.027236	-0.035731	-0.024152	-0.00911
2018-04-09	0.008802	0.003015	-0.000505	0.002868	0.007539	0.012441	-0.005372	-0.067330	0.000497	0.002745	0.005294	0.010198	0.00139

In [160…
```python
Best_returns_daily.columns
```

Out[160…
```
Index(['AMGN', 'ANET', 'BIIB', 'BIO', 'BKNG', 'DKS', 'EBAY', 'EXEL', 'FFIV',
       'HOLX', 'HPQ', 'INTU', 'LOGI', 'LPX', 'LRCX', 'META', 'MU', 'NTAP',
       'NVDA', 'REGN', 'RHI', 'STLD', 'SWKS', 'TPL', 'TXN', 'VRTX', 'WBD',
       'WSM'],
      dtype='object', name='ticker')
```

In [27]:
```python
Worst_returns_daily.columns
```

Out[27]:
```
Index(['ABC', 'ALK', 'ARMK', 'CNP', 'DRI', 'EIX', 'EXPE', 'FANG', 'FCX', 'GME',
       'HES', 'LNG', 'LYV', 'NCLH', 'NTNX', 'NTRA', 'NVAX', 'PCG', 'PLUG',
       'RUN', 'SLB', 'SNAP', 'TRGP', 'TSLA', 'VST', 'W', 'WOLF', 'WYNN'],
      dtype='object', name='ticker')
```

In [14]:

```
import pandas as pd
import numpy as np
import math
import plotly.express as px
import plotly.graph_objs as go
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pypfopt
from pypfopt import risk_models, expected_returns, plotting
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import objective_functions
from pypfopt.plotting import plot_weights
from scipy.stats import ttest_ind
import copy
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import SimpleRNN,LSTM, Dense, Dropout
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import datetime as dt
from datetime import datetime
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, TensorBoarc
import random
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from pandas_datareader.famafrench import get_available_datasets
import pandas_datareader.data as web
```

In [15]:

```
df_w = pd.read_csv("W.csv", index_col="Date")
df_b = pd.read_csv("B.csv", index_col="Date")
len(get_available_datasets())
FF = web.DataReader('F-F_Research_Data_5_Factors_2x3_daily', 'famafrench', start='2017-12-
print(FF['DESCR'])
data = FF[0]
data = data.dropna()
data = data/100
FF_start_date = df_w.first_valid_index()
FF_end_date = df_w.last_valid_index()

FF_data = pd.DataFrame(data[FF_start_date:FF_end_date])
```

```
F-F Research Data 5 Factors 2x3 daily
-------------------------------------

This file was created by CMPT_ME_BEME_OP_INV_RETS_DAILY using the 202205 CRSP database. Th
e 1-month TBill return is from Ibbotson and Associates, Inc.

  0 : (1111 rows x 6 cols)
```

In [16]:

```
data=list(FF_data.index)
df_w= df_w[:-10]
df_b= df_b[:-10]

test=FF_data.reset_index()
test=test.drop(['Date'],axis=1)
test_2=df_b.reset_index()
Data=pd.concat([test,test_2],axis=1)
Data.set_index('Date')
```

Out[16]:

Date	Mkt-RF	SMB	HML	RMW	CMA	RF	AMGN	ANET	BIIB	BIO	...	NVDA
2018-04-03	0.0124	-0.0003	0.0018	0.0026	0.0017	0.00007	0.013068	0.000000	-0.001536	0.006993	...	0.019434
2018-04-04	0.0117	0.0034	-0.0031	0.0006	-0.0012	0.00007	0.031506	0.021506	0.022216	0.005660	...	0.003958
2018-04-05	0.0075	0.0005	0.0047	0.0010	0.0027	0.00007	-0.008760	0.014597	-0.027239	-0.012175	...	-0.021482
2018-04-06	-0.0219	0.0037	-0.0006	0.0014	-0.0004	0.00007	-0.022442	-0.015300	-0.027662	-0.013618	...	-0.032216
2018-04-09	0.0030	-0.0024	-0.0051	-0.0075	-0.0027	0.00007	0.008802	0.003015	-0.000505	0.002868	...	0.005414
...	...	...	...	...	...	...	...	...	...	...	...	...
2022-05-24	-0.0123	-0.0046	0.0184	0.0141	0.0115	0.00001	0.011444	-0.025513	0.015747	-0.003879	...	-0.044029
2022-05-25	0.0122	0.0074	0.0021	-0.0053	0.0000	0.00001	0.004446	0.012639	-0.000889	-0.019586	...	0.050823
2022-05-26	0.0218	0.0002	-0.0063	-0.0014	-0.0031	0.00001	0.000158	0.027340	0.009192	0.027312	...	0.051605
2022-05-27	0.0258	0.0002	-0.0130	-0.0063	-0.0025	0.00001	0.008733	0.020827	0.009695	0.047449	...	0.053778
2022-05-31	-0.0071	-0.0039	0.0044	0.0079	-0.0024	0.00001	0.005798	-0.033910	-0.030068	-0.017286	...	-0.007389

1049 rows × 34 columns

In [17]:

```
sc = StandardScaler()
def set_seeds(seed=100):
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(100)

data = Data
cols = list(data.columns)
```

Create one lagged value for each index

In [18]:

```
def lag(lag):
    cools = []
    for i in range(len(data.columns)):
        col = f'{cols[i]}_{lag}'
        data[col] = data[cols[i]].shift(lag)
        cools.append(col)
```

In [19]:

```
lag(1)
data.dropna(inplace=True)
data=data.iloc[:,6:]
data=data.set_index('Date')
data.dropna(inplace=True)
data=data.drop(['Date_1'],axis=1)
```

In [20]:

```
data[:756]
```

Out[20]:

Date	AMGN	ANET	BIIB	BIO	BKNG	DKS	EBAY	EXEL	FFIV	HOLX	...
2018-04-04	0.031506	0.021506	0.022216	0.005660	0.001664	0.039337	0.004544	0.048402	0.007517	0.013322	...
2018-04-05	-0.008760	0.014597	-0.027239	-0.012175	0.009696	0.012237	0.006032	-0.009413	-0.001185	0.004830	...
2018-04-06	-0.022442	-0.015300	-0.027662	-0.013618	-0.027584	-0.050886	-0.023482	-0.045701	-0.017663	-0.027236	...
2018-04-09	0.008802	0.003015	-0.000505	0.002868	0.007539	0.012441	-0.005372	-0.067330	0.000497	0.002745	...
2018-04-10	0.020693	0.029626	0.029396	0.021324	0.014082	0.011410	0.018261	0.076767	0.029550	0.013140	...
...	...	...	...	...	...	...	...	...	...	...	...
2021-03-29	0.008305	-0.011501	-0.001988	-0.014777	0.011977	-0.036587	-0.016110	-0.026884	-0.007888	0.010624	...
2021-03-30	-0.020435	-0.028691	-0.000254	-0.010333	-0.007004	0.028947	0.007853	-0.004982	-0.005508	-0.004447	...
2021-03-31	-0.003764	0.010132	0.013550	0.012443	-0.002163	-0.017293	0.015252	0.028220	0.004768	0.006904	...
2021-04-01	0.001447	0.020378	-0.003718	0.015302	0.022585	0.040972	0.030536	0.023462	0.011936	-0.000538	...
2021-04-05	0.011438	0.011622	-0.002978	0.004984	0.011215	0.016147	0.009190	0.015571	0.019753	0.000269	...

756 rows × 62 columns

In [21]:

```
X_train = []
y_train = []

n_future = 30
n_past = 60
n_cols = 11
set_seeds()


MSE_all = []
for j in range(len(data.columns[:28])):
    MSE = []
    for i in range(n_past, len(data[:756]) - n_future +1,30):
        X_train = data[:756].iloc[i-n_past:i,28:]
        y_train = data[:756].iloc[i-n_past:i,j]
        X_test = data[:756].iloc[i:i+n_future,28:]
        y_test = data[:756].iloc[i:i+n_future,j]
        X_train_scaled = sc.fit_transform(X_train)
        X_test_scaled = sc.fit_transform(X_test)
        model=MLPRegressor(hidden_layer_sizes=(10,5,1),activation="logistic" , max_iter=2(
        reg = model.fit(X_train_scaled, y_train)
        y_pred=reg.predict(X_test_scaled)
        MSE.append(mean_squared_error(y_pred, y_test))
    MSE_all.append(np.mean(np.array(MSE)))
```

The model below is my optimal version:

In [22]:
```
np.mean(np.array(MSE_all))
```

Out[22]:
```
0.0023323069783450366
```

In [24]:
```
df_test = pd.DataFrame(index = data.columns[0:28])
df_test['Avg MSE'] = MSE_all
```

In [26]:
```
df_test
```

Out[26]:

	Avg MSE
AMGN	0.001334
ANET	0.002535
BIIB	0.002642
BIO	0.001742
BKNG	0.002175
DKS	0.002972
EBAY	0.001638
EXEL	0.002750
FFIV	0.001621
HOLX	0.001872
HPQ	0.002239
INTU	0.002017
LOGI	0.002421
LPX	0.002798
LRCX	0.001988
META	0.002530
MU	0.003015
NTAP	0.002521
NVDA	0.003369
REGN	0.001977
RHI	0.001650
STLD	0.002750
SWKS	0.002694
TPL	0.002674
TXN	0.002517
VRTX	0.001808
WBD	0.002836

	Avg MSE
WSM	0.002222

In [12]:

```
y_test_all  = pd.DataFrame()
```

In [13]:

```
data[756:906]
```

Out[13]:

Date	AMGN	ANET	BIIB	BIO	BKNG	DKS	EBAY	EXEL	FFIV	HOLX	...
2021-04-06	-0.011586	-0.012027	-0.022024	0.008150	0.005321	0.018250	-0.018370	-0.006388	0.000325	0.004841	...
2021-04-07	-0.003934	-0.000169	-0.013762	-0.018569	-0.004690	-0.008169	-0.012156	0.003858	-0.020664	-0.024358	...
2021-04-08	-0.004998	0.011136	-0.008544	0.027036	0.007197	0.006761	0.006962	-0.002562	0.003082	0.009328	...
2021-04-09	0.008385	0.013828	0.009182	0.017459	0.009090	0.003175	0.004502	-0.010274	-0.005342	0.000680	...
2021-04-12	-0.000040	-0.004893	-0.020025	0.002357	-0.016738	0.016310	0.002721	-0.003893	-0.015303	0.010729	...
...	...	...	...	...	...	...	...	...	...	...	...
2021-10-29	-0.000965	0.006506	0.006834	0.007429	-0.003700	0.006809	0.059522	-0.011489	-0.011053	0.008946	...
2021-11-01	0.013625	-0.002734	0.020324	-0.013238	0.028466	-0.027292	-0.005474	0.008833	0.018802	-0.008594	...
2021-11-02	0.021307	0.203893	0.001507	-0.006478	-0.014403	0.028720	-0.018349	0.008295	0.034864	-0.024353	...
2021-11-03	0.018062	0.044581	0.025981	0.000988	-0.007539	0.058251	0.007343	-0.120658	-0.002875	0.019743	...
2021-11-04	-0.014670	0.019580	0.002933	0.004321	0.000690	-0.015206	0.011001	-0.016112	0.002928	-0.006638	...

150 rows × 62 columns

In [27]:

```
X_train = data[756:906].drop(data.columns[0:28],axis=1)
X_test = data[906:].drop(data.columns[0:28],axis=1)
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.fit_transform(X_test)

model = MLPRegressor(hidden_layer_sizes=(10,5,1),
                   activation="logistic",
                   max_iter=2000,
                   learning_rate_init=.01)

y_test_pall = pd.DataFrame()
MSE_all  = []
```

In [28]:

```
for i in range(len(data.columns[0:28])):
    y_train = data.iloc[756:906, i]
```

```
    y_test = data.iloc[906:, i]
    reg = model.fit(X_train_scaled, y_train)
    y_pred = reg.predict(X_test_scaled)
    col_name = y_train.name + '_pred'
    y_test = y_test.to_frame()
    y_test[col_name] = y_pred
    mse = mean_squared_error(y_test[y_train.name], y_test[col_name])
    MSE_all.append(mse)
    y_test = y_test[col_name]
    y_test_pall = pd.concat([y_test_pall,y_test], axis=1)
```

In [29]:

```
cov= y_test_pall.copy()
```

In [30]:

```
df_8 = pd.DataFrame(index =y_test_pall.columns)
df_8['Avg Daily Return'] = y_test_pall.mean()
df_8['Volatility'] = y_test_pall.std()
df_8['Sharpe Ratio'] = y_test_pall.mean()/y_test_pall.std()
df_8['MSE'] = MSE_all
df_8=df_8.sort_values(by=['Avg Daily Return','Sharpe Ratio'],ascending=False)

df_8= df_8[:9]
df_8
```

Out[30]:

	Avg Daily Return	Volatility	Sharpe Ratio	MSE
WSM_pred	0.066670	0.009603	6.942269	0.006151
BIIB_pred	0.053843	0.007978	6.748730	0.003669
TPL_pred	0.050324	0.013972	3.601862	0.003433
LRCX_pred	0.034508	0.010050	3.433561	0.002310
RHI_pred	0.033408	0.001680	19.886826	0.001649
HPQ_pred	0.033382	0.006759	4.939216	0.001807
BIO_pred	0.025382	0.001500	16.919366	0.001336
NTAP_pred	0.024630	0.002183	11.285036	0.001112
MU_pred	0.022125	0.000398	55.562861	0.001473

In [18]:

```
col = df_8.index
cov = y_test_pall[col]
```

In [19]:

```
y_test_pall
```

Out[19]:

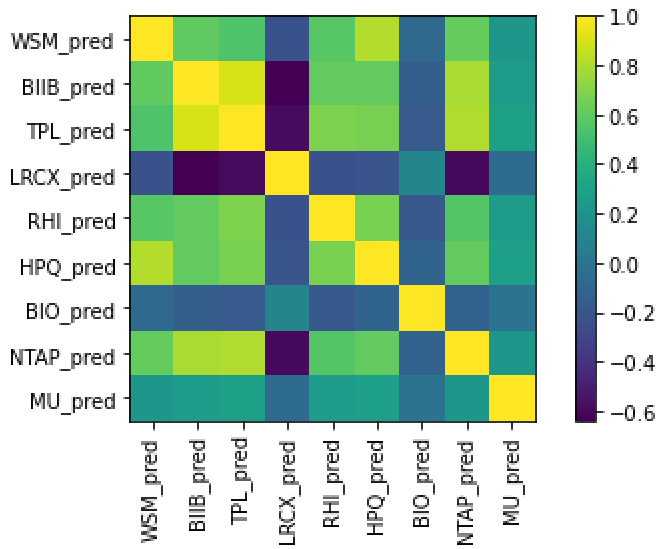
	AMGN_pred	ANET_pred	BIIB_pred	BIO_pred	BKNG_pred	DKS_pred	EBAY_pred	EXEL_pred	FFIV_pred	HC
2021-11-05	-0.028505	-0.047185	0.061203	0.023672	-0.042352	-0.056006	-0.036237	-0.053806	0.008213	-
2021-11-08	-0.028539	-0.041021	0.052209	0.023411	-0.042703	-0.074306	-0.040861	-0.069587	0.006396	-
2021-11-09	-0.028448	-0.049528	0.054247	0.027990	-0.042840	-0.080687	-0.036867	-0.055905	0.011750	-
2021-11-10	-0.028652	-0.044299	0.056103	0.026184	-0.042858	-0.066562	-0.038069	-0.059360	0.006887	-

	AMGN_pred	ANET_pred	BIIB_pred	BIO_pred	BKNG_pred	DKS_pred	EBAY_pred	EXEL_pred	FFIV_pred	HC
2021-11-11	-0.028097	-0.052459	0.048334	0.025869	-0.042913	-0.081599	-0.036766	-0.048808	0.009186	-
...	...	...	...	...	...	...	...	...	...	
2022-05-24	-0.028321	-0.047852	0.061728	0.026882	-0.042702	-0.079585	-0.035708	-0.066047	0.007929	-
2022-05-25	-0.028031	-0.055512	0.048014	0.023584	-0.042965	-0.080119	-0.038390	-0.053877	0.008628	-
2022-05-26	-0.028635	-0.050908	0.064088	0.025174	-0.042640	-0.061576	-0.036899	-0.065314	0.008847	-
2022-05-27	-0.028696	-0.047445	0.063821	0.023970	-0.042627	-0.053251	-0.037073	-0.065454	0.004054	-
2022-05-31	-0.028364	-0.034439	0.064138	0.025222	-0.042290	-0.070043	-0.036053	-0.064932	0.008063	-

142 rows × 28 columns

In [28]:
```
mu = df_8['Avg Daily Return']
S = risk_models.CovarianceShrinkage(cov,returns_data=True,  frequency=252, log_returns=Fal
plotting.plot_covariance(S, plot_correlation=True);

# S = risk_models.CovarianceShrinkage(prices).ledoit_wolf()
```



In [29]:
```
mu
```

Out[29]:
```
WSM_pred     0.066670
BIIB_pred    0.053843
TPL_pred     0.050324
LRCX_pred    0.034508
RHI_pred     0.033408
HPQ_pred     0.033382
BIO_pred     0.025382
NTAP_pred    0.024630
MU_pred      0.022125
Name: Avg Daily Return, dtype: float64
```

In [ ]:
```
###################get risk free rate from kenneth french####################
len(get_available_datasets())
```

```
ds = web.DataReader('F-F_Research_Data_5_Factors_2x3_daily', 'famafrench', start='2017-12-

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 = df_w.first_valid_index()
RF_end_date = df_w.last_valid_index()

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

In [ ]:

```
df_8 = pd.DataFrame(index = y_test_pall.index )
y_temp = data.iloc[119:191,0:10]
y_temp
```

In [ ]:

```
for i in range(len(y_test_pall.columns)):
    col = y_test_pall.columns[i]
    col_dir = col + '_dir'
    col_str = col + '_str'
    df_8[col_dir] = np.where(y_test_pall[col]> 0.001, 1, -1)
    df_8[col_str] = (df_8[col_dir] * y_temp.iloc[:,i])
```

In [ ]:

```
mu = df_7['Avg Monthly Return']
S= risk_models.exp_cov(y_test_pall,returns_data=True,  frequency=12, log_returns=False)
plotting.plot_covariance(S, plot_correlation=True);
ef = EfficientFrontier(mu, S)
ef.add_objective(objective_functions.L2_reg, gamma=1)
ef.efficient_risk(0.15)
weights = ef.clean_weights()
weights
```

In [ ]:

```
S= risk_models.exp_cov(y_test_pall,returns_data=True,  frequency=12, log_returns=False)
```

In [ ]:

```
plotting.plot_covariance(S, plot_correlation=True);
```

In [58]:

```
sector_mapper = {
    "WSM_pred": "Consumer Discretionary",
    "BIIB_pred": "Biotechnology",
    "TPL_pred": "Real estate ",
    "LRCX_pred": "Semiconductor ",
    "BIO_pred": "Biotechnology ",
    "NTAP_pred": "Technology",
    "MU_pred": "Semiconductor",
    "RHI_pred": "Financial Services",
    "HPQ_pred": "Technology"
}

sector_lower = {
    "Biotechnology": 0.1,
    "Tech": 0.1 # at least 5% to tech
    # For all other sectors, it will be assumed there is no lower bound
```

```
}

sector_upper = {
    "Tech": 0.2
}
```

In [82]:

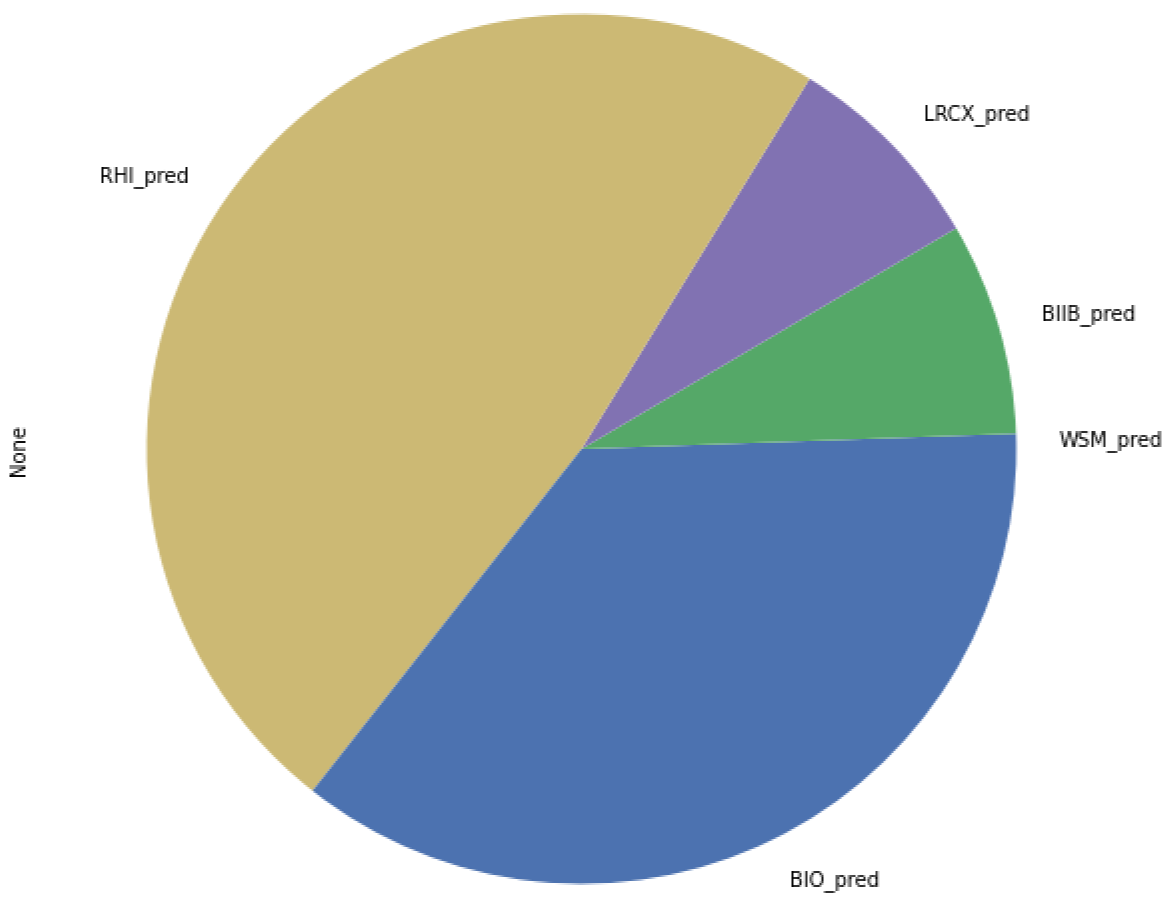
```
ef = EfficientFrontier(mu, S)  # weight_bounds automatically set to (0, 1)
# ef.add_objective(objective_functions.L2_reg, gamma=.0001)  # gamme is the tuning paramet
ef.max_sharpe()
weights = ef.clean_weights()
weights
```

Out[82]:

```
OrderedDict([('WSM_pred', 0.00563),
             ('BIIB_pred', 0.07914),
             ('TPL_pred', 0.0),
             ('LRCX_pred', 0.07737),
             ('RHI_pred', 0.48174),
             ('HPQ_pred', 0.0),
             ('BIO_pred', 0.35611),
             ('NTAP_pred', 0.0),
             ('MU_pred', 0.0)])
```

In [83]:

```
pd.Series(weights).plot.pie(figsize=(10,10));
```



In [ ]:

In [ ]:

In [ ]:
```
df_10_str = df_10.drop(columns = cols)
```

In [ ]:
```
df_10_str.sum().apply(np.exp)
```

In [ ]:
```
df_10_str.cumsum().apply(np.exp).plot(figsize=(10, 6))
```

In [ ]:

In [21]:

```
import yfinance as yf
import pandas as pd
import numpy as np
import math
import plotly.express as px
import plotly.graph_objs as go
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pypfopt
from pypfopt import risk_models, expected_returns, plotting
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import objective_functions
from pypfopt.plotting import plot_weights
from scipy.stats import ttest_ind
import copy
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import SimpleRNN,LSTM, Dense, Dropout
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import datetime as dt
from datetime import datetime
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, TensorBoard
import random
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from pandas_datareader.famafrench import get_available_datasets
import pandas_datareader.data as web
import statsmodels.api as sm
```

In [22]:

```
tickers = ["BIIB","LRCX","RHI", "WSM","BIO","^GSPC"]
```

In [23]:

```
ohlc = yf.download(tickers, start="2022-02-28")
```

```
[*********************100%***********************]  6 of 6 completed
```

In [24]:

```
prices = ohlc["Adj Close"].dropna(how="all")
daily = prices[:-23].pct_change()
daily.dropna(inplace=True)
daily['Portfolio Return']=daily['BIIB']*0.08+daily['BIO']*0.10+daily['LRCX']*0.075+daily['
Return=list(daily['Portfolio Return'])
```

In [25]:

```
daily.head(5)
```

Out[25]:

Date	BIIB	BIO	LRCX	RHI	WSM	^GSPC	Portfolio Return
2022-03-01	-0.003175	-0.014713	-0.037018	-0.055865	0.012771	-0.015474	-0.019785
2022-03-02	-0.012123	-0.007896	0.024881	0.034428	0.020721	0.018643	0.010955
2022-03-03	0.016555	-0.005132	-0.018971	0.005192	0.012755	-0.005255	0.001506
2022-03-04	-0.008427	-0.043040	-0.032621	-0.009061	0.001055	-0.007934	-0.009954
2022-03-07	-0.017522	-0.055584	-0.069668	-0.055290	-0.068436	-0.029518	-0.031365

In [26]:

```
len(get_available_datasets())
FF = web.DataReader('F-F_Research_Data_5_Factors_2x3_daily', 'famafrench', start='2022-03-
print(FF['DESCR'])
data = FF[0]
data = data.dropna()
data = data/100
data['Port_Ret']=Return
FF_5 = pd.DataFrame(data)
FF_5['RP-Rf']=FF_5['Port_Ret']-FF_5['RF']
X = FF_5[['Mkt-RF','SMB','HML','RMW','CMA']]
Y = FF_5['RP-Rf']
X = sm.add_constant(X)
```

```
F-F Research Data 5 Factors 2x3 daily
-------------------------------------

This file was created by CMPT_ME_BEME_OP_INV_RETS_DAILY using the 202205 CRSP database. Th
e 1-month TBill return is from Ibbotson and Associates, Inc.

  0 : (63 rows x 6 cols)
C:\Users\DELL\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning:
In a future version of pandas all arguments of concat except for the argument 'objs' will
be keyword-only
  x = pd.concat(x[::order], 1)
```

In [27]:

```
FF_5.head(5)
```

Out[27]:

Date	Mkt-RF	SMB	HML	RMW	CMA	RF	Port_Ret	RP-Rf
2022-03-01	-0.0156	-0.0028	-0.0083	-0.0037	0.0059	0.0	-0.019785	-0.019785
2022-03-02	0.0181	0.0062	0.0133	0.0047	0.0042	0.0	0.010955	0.010955
2022-03-03	-0.0088	-0.0021	0.0151	0.0124	0.0120	0.0	0.001506	0.001506
2022-03-04	-0.0110	-0.0049	0.0068	0.0041	0.0121	0.0	-0.009954	-0.009954
2022-03-07	-0.0312	0.0071	0.0090	-0.0069	0.0136	0.0	-0.031365	-0.031365

In [28]:

```
model = sm.OLS(Y, X).fit()
predictions = model.predict(X)
print_model = model.summary()
print(print_model)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  RP-Rf   R-squared:                       0.841
Model:                            OLS   Adj. R-squared:                  0.827
Method:                 Least Squares   F-statistic:                     60.18
Date:                Mon, 04 Jul 2022   Prob (F-statistic):           1.73e-21
Time:                        16:46:28   Log-Likelihood:                 241.12
No. Observations:                  63   AIC:                            -470.2
Df Residuals:                      57   BIC:                            -457.4
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0004      0.001     -0.529      0.599      -0.002       0.001
Mkt-RF         0.6520      0.053     12.402      0.000       0.547       0.757
```

```
SMB              0.2477      0.147      1.685      0.097      -0.047       0.542
HML              0.0213      0.111      0.192      0.849      -0.201       0.243
RMW              0.2640      0.106      2.501      0.015       0.053       0.475
CMA             -0.2886      0.161     -1.788      0.079      -0.612       0.035
==============================================================================
Omnibus:                       10.524   Durbin-Watson:                   2.222
Prob(Omnibus):                  0.005   Jarque-Bera (JB):               18.836
Skew:                          -0.463   Prob(JB):                     8.13e-05
Kurtosis:                       5.513   Cond. No.                         303.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
d.
```

In [ ]: