Course: EE590 Project Report

Name: Linyin Yang
Student ID: 4434502377
Email: linyinya@usc.edu
Term: Spring 2022
Project Title: Comparing Portfolio Optimizations on Stocks Investments
Date: April 20, 2022

Executive Summary

This project mainly discusses how individual or non-professional investors can distribute the assets in a portfolio in general and compares several stock optimization methods to help make investment decisions. The ultimate purpose of any investment activities must be to earn as many returns as possible. However, different investors have different abilities to tolerate the investment risks. As a result, giving investment advisement should depend on various conditions and goals. Commonly, there are three goals to be considered: maximizing the Sharpe ratio, minimizing the volatility, and simply maximizing means.

To simulate and experiment with portfolio investment activities, I suppose that I initially have $1,000,000 to invest in the stock market. I selected a ten-stock portfolio and tried to design methods to assign weights to the stocks. The only data necessary for the analysis is the historical price data. I obtained other information such as daily returns and excess daily returns from the prices data that are also critical to the methods implementations. The historical data is time-series data and, therefore, is time-sensitive. Consequently, I believe that it is useless to consider data from years or decades ago. In other words, only around five years of data meet the requirements.

Using the historical data, I developed three different models. Two of them were improved based on the mean-variance analysis method. The significant difference between the two methods is the method of estimating the expected return. While the first method adapted some linear regression ideas to calculate the expected return, the second method considered giving more recent data higher weights. After some basic calculations, I computed the results, i.e., the weights assigned to stocks, conditioned by different investment goals. During the testifying process on the first method, I observed that solely maximizing the investment returns never generates satisfying results. So, I dropped this goal during the implementation of the second method. The results of the first two methods prefer the first method over the second. The first method only generated an average-level return portfolio, and it has very little room for improvement. The second method's output depicted different result patterns of the two investment goals, giving the investors ideas of balancing high returns and high volatilities. Yet, there are still considerable limitations on improvements. The third method used a completely different view. The model is built based on deep learning techniques. A modified recurrent neural network architecture is added to the model. The model is also trained by the historical price data of each stock and outputs some estimations of the future stock price. This method is a lot more flexible than the previous two methods regarding the number of parameters to tune and the room for improvements.

All the models have ready-to-run spreadsheets or python codes with detailed instructions. All investors interested in stock portfolio investment could easily use the models as additional tools. They could, to some extent, help with making portfolio investment decisions. However, the stock market is a very dynamic environment. Even though the project has validated its results and gave recommendations to the investors, the results could not guarantee to work well under every circumstance in the future. Therefore, the results are only auxiliary outcomes, and investors should be cautious.

1. Introduction

More and more people globally started to manage their wealth through personal investment in recent years. For individual investors, buying and selling shares in the stock market is the easiest way to invest with few limitations. As a result, "how to distribute assets in different stocks" or "how to form a stock portfolio" became frequent questions that individual investors would ask. This project will discuss this topic in depth through different aspects.

The primary purpose for individuals who invest in the stock market is to earn as much return as possible. A Gallup report asserts that around 56% of American adults are personally invested in the US stock market in 2021[1]. This result means that more than half of adults in the United States own shares in those publicly traded companies and are willing to make profits from the market, not to mention millions of foreign investors who own shares in the US stock market. Therefore, the earlier problem is considerably practical and may benefit millions of people. The Gallup report also shows that this percentage was about 6-7% higher historically. It remains over 60% for an extended period and has once reached over 65%. The decline appeared in 2008 after witnessing the Great Recession. Many investors turned to real estate, gold, and bonds to seek more stable long-term investments [2]. This phenomenon could imply that volatility is one primary concern for the stock market investors while maximizing their returns. Typically, investors expect higher volatility with a higher return. Investors would choose different investment strategies to achieve their investment expectations depending on their risk tolerances. There could be different methods to approach the investor's final goal for each strategy. For example, there are different ways of calculating parameters such as expected return and the betas. These additional calculations directly lead to different compositions of the stock portfolio.

The primary purpose of this project is to answer how individual investors can decide and distribute their assets in a stock portfolio in general. I followed several different ideas to solve this problem. More specifically, this project considered the problem with three different optimization goals for optimizing a ten stocks portfolio: maximizing the return, minimizing the volatility, and maximizing the Sharpe ratio. The first two optimization methods are straightforward, as stated. In the last approach, the Sharpe ratio is the ratio that measures the risk-adjusted return. It tells the average return earned by an investor per excess unit of volatility. I developed a model to recommend the distribution or give an idea of how to distribute assets on stocks to achieve the goals. One major approach to building the model is the mean-variance optimization method. My project is majorly developed based on the basic mean-variance method and its improved variant. I also tried machine learning techniques for optimizing the distributions.

The project results show that neither maximizing the Sharpe ratio nor minimizing volatility gives satisfying optimization results for the basic mean-variance method. The difference between the result of these two goals was not significant. However, when I took the exponential moving average (EMA) into account, the difference between these two methods became outstanding.

2. Problem Statement

The main problem of this project is to form a ten stocks portfolio and optimize it with different methods.

There are three phases in the project. The first phase is to form the portfolio. Initially, I supposed that I had $1,000,000 to invest. I hand-picked ten stocks to form the portfolio and experimented based on their historical data. The idea of forming the portfolio for this project is to form a good portfolio for a common investor. It is typical for individual investors, especially green-handed investors, to choose stocks of companies they believe in having promising futures. Under the current market environment, big-tech companies' stocks are selected with high priorities for the following reasons. The technology industry is one of the current leading and continuously developing industries. It is also less impacted by the COVID-19 period than companies in other industries, so outlier data points in the historical data might be less if we think statistically. Therefore, I decided to form my portfolio mostly with big-tech companies and some security stocks in the financial industry. The ten stocks I chose are: Apple Inc. (AAPL), Amazon.com, Inc.(AMZN), Bank of America Corporation(BAC), Meta Platforms, Inc.(FB), Alphabet Inc.(GOOG), JPMorgan Chase&Co.(JPM), Microsoft Corporation(MSFT), Netflix Inc.(NFLX), NVIDIA Corporation(NVDA), Tesla, Inc.(TSLA). The advantage of this portfolio is that it follows the market trends and is suitable and comparably reliable for green hands. The target group of this project result is anyone interested in investing in the stock. One potential problem for this portfolio is that the historical data from years ago (e.g., 4 or 5 years ago) might not be helpful information in analyzing future returns since the tech companies tend to be fast-growing.

The second phase is to train and validate models for analyzing historical data and recommend optimal weights of assets distributions. As the portfolio is formed, the rest of the problem is how to optimize it. This phase contains experiments and improvements repeatedly. Predicting the market is a challenging task. The historical price data of ten stocks are pretty time-sensitive. Also, many factors and emergency events might affect the market massively. Therefore, today's model may not even guarantee a good performance next quarter. So, my project's goal is to develop the methods of optimizations, not the very best model or a promising model.

For the historical data of each stock, I chose two 5-year periods as training data, one group from 2014 to 2018, the other from 2016 to 2020. I also obtained data from 2021 until now as testing data. A commonly used method to analyze the historical stock data is the mean-variance analysis. I preprocess the historical price data downloaded from Yahoo Finance for calculation needs. First, I turned the daily price data into daily return data. Then, I subtract the risk-free rate of that day to get the daily excess return.

The third phase is to test and analyze the performance and result of the optimized portfolio. My expected result is to get models that can provide recommendations for distributing their assets. The recommendations should be reasonable, easy to understand, and have good adaptivity and extendibility.

3. Approaches

The mean-variance analysis is a frequently used method to weigh investing risk against investing return [3]. The method is very suitable for achieving the project goals. It is initially designed for finding either the highest return at a given risk level or the lowest return at a given reward level, while it is easy to modify the method for maximizing the Sharpe ratio. The two major components that need calculations in the mean-variance analysis are expected return and variance. The expected return expresses the estimated return on the portfolio investment. Variance is the statistical term that measures how far each daily return is from the average return. Since ten stocks form the portfolio, the covariance between each pair of stocks is also essential to calculating and building a variance-covariance matrix. The variance-covariance matrix is directly related to volatility. After calculating the expected returns and variances, I applied the Solver tool in Excel to calculate portfolio weights. One main problem that needs further discussion here is the methods of calculating the expected returns.

It is natural to first think of using the average historical return as the expected return. The clear advantage of this method is its simplicity in understanding and calculating. Yet the problem with this method is that an average might not accurately represent the expected future return, especially in five years.

Therefore, I chose an alternative approach to calculate the expected return by implementing a single-factor model in excel for estimation. This approach is derived from the Capital Asset Pricing Model (CAPM). The CAPM expected return compensated for the risk and the time value of money [4]. The formula to calculate CAPM is: $E = R_f + \beta * E[R_m - R_f]$, where "$E$" is the CAPM expected return, "R_f" is the risk-free rate, "β" is the measure of additional risk added to the portfolio by this investment, and "$E[R_m - R_f]$" represents the market risk premium. Therefore, this method requires some additional factors such as risk-free rate, market excess returns, and others to calculate the mean and standard deviation of the excess returns on the market portfolio. This method is intuitively superior to the previous one because it considers the impacts of market changes when calculating the expected returns. The detailed implementation is described in the following paragraph.

First, I calculated the mean and standard deviation (SD) of the excess returns on the market portfolio. I used the built-in *AVERAGE()* and *STDEV()* functions to calculate these two factors correspondingly. I also calculated a reasonable risk-free rate for the data. In this project, the risk-free rate is equal to the average risk-free rate in the final year of the time interval of historical data. For example, if my data is from the start of 2014 to the end of 2018, then the risk-free rate is the average risk-free rate in 2018. Historical data of market excess returns and risk-free rates are found in the Kenneth R. French Library [5]. Theoretically, when estimating the expected excess return for each stock with market portfolio excess returns given, we think of a single index regression model. The parameters needed are the intercepts and slopes of those regression lines. In finance, the intercepts and slopes can be explained as alphas and betas, where alphas show how well stocks performed compare to the market and betas show how volatile are the stocks. One critical condition my estimation is: for the CAPM to hold, the intercept, or alpha,

must be zero because the CAPM assumes that the portfolio is similar to the market, so that, the expected values of alphas are zero. This assumption is also verified as the intercepts of the regression lines are very close to zero. Therefore, I only computed the betas of the ten stocks in the portfolio using built-in function *SLOPE()* in excel. After the above calculations, I obtained the expected excess returns of ten stocks. Adding the risk-free rate to each expected excess return, I got the CAPM expected returns. Then, I started to calculate the variance-covariance matrix. For the "regression line", there is also a standard error term often called epsilon (ϵ). This parameter is calculated by using the *STEYX()* function in excel for each stock. According to the single index model, the variance of each stock is $Var(i) = \beta_i^2 * SD(RMRF)^2 + Var(\epsilon)$, where these parameters are all calculated earlier. The standard deviation of each stock is just the square root of the variance. The covariance between two stocks is calculated by: $Cov(i,j) = \beta_i * \beta_j * SD(RMRF)^2$. Then I built a variance-covariance matrix between pairs of stocks. Next, I converted the variance-covariance matrix into the correlation matrix shown in figure 1 by $Corr(i,j) = \frac{\text{Cov(i,j)}}{\text{SD(i)}*\text{SD(j)}}$.

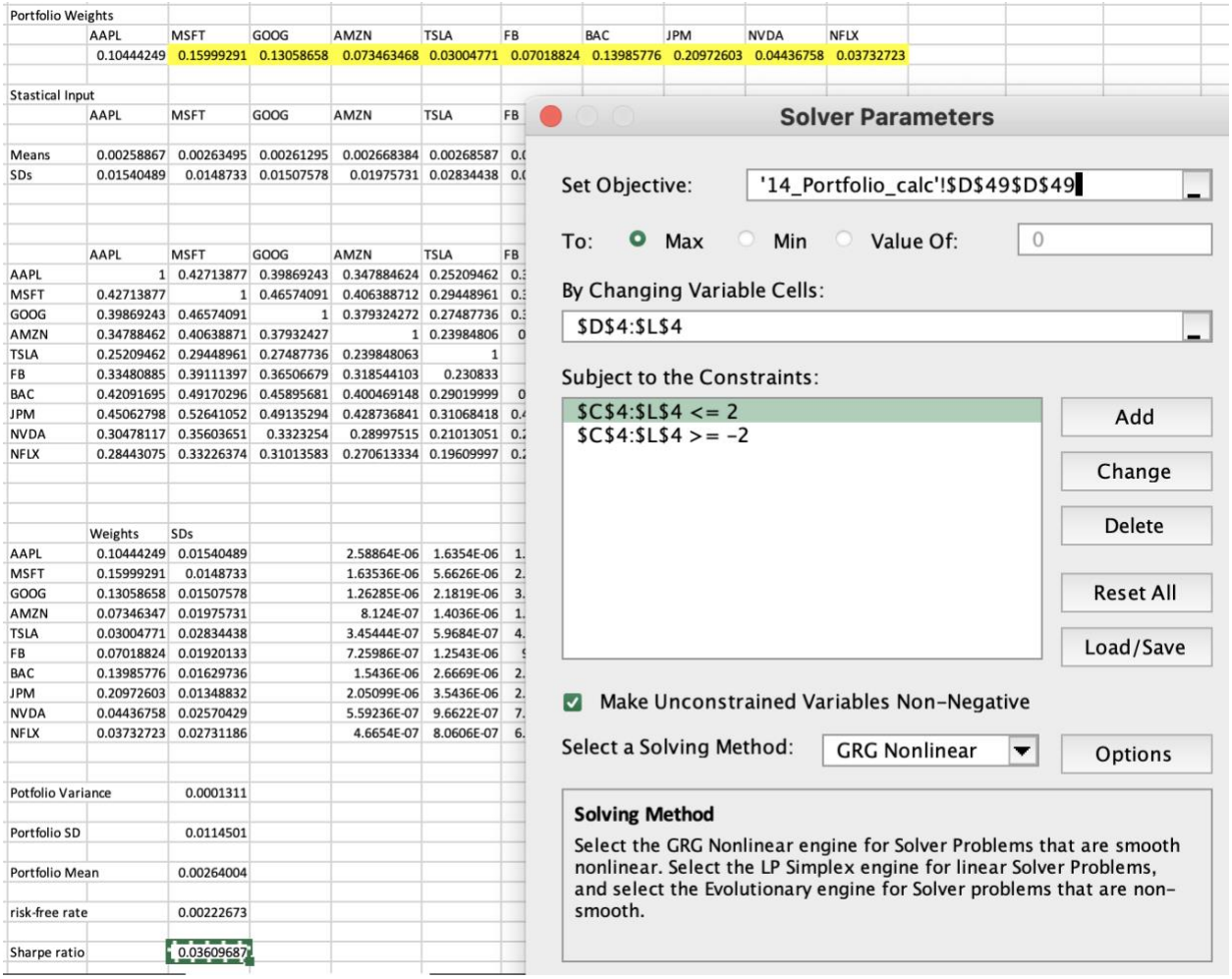
Correlation Matrix												
			AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
		betas	0.01540489	0.0148733	0.015076	0.019757	0.028344	0.019201	0.016297	0.013488	0.025704	0.027312
	betas											
AAPL	0.015405		1	0.42713877	0.398692	0.347885	0.252095	0.334809	0.420917	0.450628	0.304781	0.284431
MSFT	0.014873		0.42713877	1	0.465741	0.406389	0.29449	0.391114	0.491703	0.526411	0.356037	0.332264
GOOG	0.015076		0.39869243	0.46574091	1	0.379324	0.274877	0.365067	0.458957	0.491353	0.332325	0.310136
AMZN	0.019757		0.34788462	0.40638871	0.379324	1	0.239848	0.318544	0.400469	0.428737	0.289975	0.270613
TSLA	0.028344		0.25209462	0.29448961	0.274877	0.239848	1	0.230833	0.2902	0.310684	0.210131	0.1961
FB	0.019201		0.33480885	0.39111397	0.365067	0.318544	0.230833	1	0.385417	0.412622	0.279076	0.260442
BAC	0.016297		0.42091695	0.49170296	0.458957	0.400469	0.2902	0.385417	1	0.518743	0.35085	0.327424
JPM	0.013488		0.45062798	0.52641052	0.491353	0.428737	0.310684	0.412622	0.518743	1	0.375616	0.350536
NVDA	0.025704		0.30478117	0.35603651	0.332325	0.289975	0.210131	0.279076	0.35085	0.375616	1	0.237084
NFLX	0.027312		0.28443075	0.33226374	0.310136	0.270613	0.1961	0.260442	0.327424	0.350536	0.237084	1

Figure 1: Correlation Matrix calculated for 2014-2018 historical data on the portfolio.

The above part describes the preparation of mean-variance analysis. For the analysis, our input data are the CAPM expected returns, standard deviations, and the correlation matrix. I initialized the weights of each stock in the portfolio to 0.1. To ensure that these weights add up to 1, I set the weight of the first stock $w_1 = 1 - \sum w_i$, for $i \neq 1$. The last step of calculation is to compute the portfolio variance between two assets (same two assets also count). The formula for calculating the portfolio variance (Port_Var) between two assets is $Port\_Var(i,j) = w_i * w_j * SD(i) * SD(j) * Corr(i,j)$. The total portfolio variance is the sum of all pairs of variances. I took square root of the portfolio variance to get the portfolio standard deviation (σ), which is often seen as the volatility of the portfolio. The portfolio mean (μ) is weighted sum of each stock. The Sharpe Ratio is then calculated by formula $SharpeRatio = \frac{\mu - r_f}{\sigma}$.

The final step of this implementation for mean-variance analysis is to use the built-in Solver tool in Excel to achieve our preset goals. As shown in figure 2, I could select one of the parameters and select whether to minimize or maximize it by changing the nine other weights (in yellow). Then an optimal set of weights from this model will be out. I could also add constraints to the weights. For example, if I want to build a long position only portfolio, I can set the weights

to all be greater than 0 and less than 1. In this project, I set the weights to be in the interval of [-2, 2].



Portfolio Weights										
	AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
	0.10444249	0.15999291	0.13058658	0.073463468	0.03004771	0.07018824	0.13985776	0.20972603	0.04436758	0.03732723

Stastical Input						
	AAPL	MSFT	GOOG	AMZN	TSLA	FB
Means	0.00258867	0.00263495	0.00261295	0.002668384	0.00268587	0.0
SDs	0.01540489	0.0148733	0.01507578	0.01975731	0.02834438	0.0

	AAPL	MSFT	GOOG	AMZN	TSLA	FB
AAPL	1	0.42713877	0.39869243	0.347884624	0.25209462	0.3
MSFT	0.42713877	1	0.46574091	0.406388712	0.29448961	0.3
GOOG	0.39869243	0.46574091	1	0.379324272	0.27487736	0.3
AMZN	0.34788462	0.40638871	0.37932427	1	0.23984806	0
TSLA	0.25209462	0.29448961	0.27487736	0.239848063	1	
FB	0.33480885	0.39111397	0.36506679	0.318544103	0.230833	
BAC	0.42091695	0.49170296	0.45895681	0.400469148	0.29019999	0
JPM	0.45062798	0.52641052	0.49135294	0.428736841	0.31068418	0.4
NVDA	0.30478117	0.35603651	0.3323254	0.28997515	0.21013051	0.2
NFLX	0.28443075	0.33226374	0.31013583	0.270613334	0.19609997	0.2

	Weights	SDs				
AAPL	0.10444249	0.01540489		2.58864E-06	1.6354E-06	1.
MSFT	0.15999291	0.0148733		1.63536E-06	5.6626E-06	2.
GOOG	0.13058658	0.01507578		1.26285E-06	2.1819E-06	3.
AMZN	0.07346347	0.01975731		8.124E-07	1.4036E-06	1.
TSLA	0.03004771	0.02834438		3.45444E-07	5.9684E-07	4.
FB	0.07018824	0.01920133		7.25986E-07	1.2543E-06	9
BAC	0.13985776	0.01629736		1.5436E-06	2.6669E-06	2.
JPM	0.20972603	0.01348832		2.05099E-06	3.5436E-06	2.
NVDA	0.04436758	0.02570429		5.59236E-07	9.6622E-07	7.
NFLX	0.03732723	0.02731186		4.6654E-07	8.0606E-07	6.

Potfolio Variance		0.0001311
Portfolio SD		0.0114501
Portfolio Mean		0.00264004
risk-free rate		0.00222673
Sharpe ratio		0.03609687

Figure 2: Using Solver tool to calculate corresponding portfolio weights to different goals.

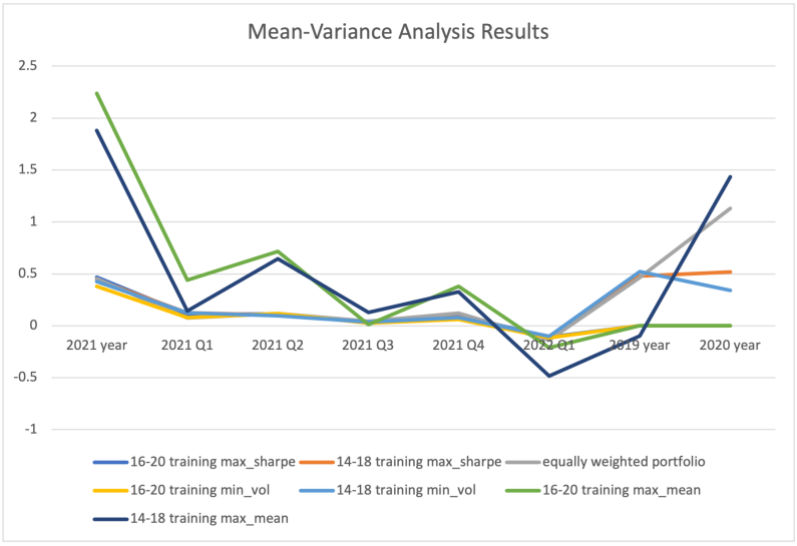


Figure 3: The returns of portfolio with different training historical data.

The results of this implementation of mean-variance analysis are displayed in figure 3. I wrote a python script to test for the output portfolio weights, and the testing process is designed as the following. First, I set my initial fund to be $1000000 and decided the period of this investment. In my project, I chose to test with several three-month periods from 2021 until now and test with longer terms (1 year). One significant observation is that maximizing the portfolio's mean produces very volatile results (green and navy-blue lines in figure 3). It also makes very polarized weights: only 2 to 3 best or worst performed stocks in the historical data are assigned weights. Therefore, this goal is not very helpful for portfolio optimization. To maximize the Sharpe ratio and minimize volatility, they both produce similar and stable results. But the problem here is that they produces results no better than an equally weighted portfolio. These results show that this implementation of mean-variance analysis may not be very efficient and optimized. I think that this model is not optimal because the estimations of expected returns of stocks are not very accurate. Therefore, I decided to make some modifications to this estimating process.

I considered exponentially moving average (EMA) to assign exponential weights to the historical return data from the oldest to the newest, such that the latest data has the highest weights. This process could be done by Excel if I manually create some exponential weights. For experimental convenience, I used the *PyPortfolioOpt* library in python to implement the mean-variance analysis using exponentially weighted means [6]. First, I read the daily return data into a data frame in python. The daily return of each stock was calculated during preprocessing. Then, I used the *pypfopt.expected_returns.ema_historical_return()* function to calculate exponentially-weighted mean of (daily) historical returns (a sample output is given in figure 4), giving higher weight to more recent data. The “span” and “frequency” are two parameters that can be tuned. The “span” means the timespan for the EMA, and the “frequency” means the number of periods in a year. Next, I used *pypfopt.risk_models.exp_cov()* function to estimate the exponentially-weighted covariance matrix. The risk-free rate is calculated in the same way as the basic mean-variance analysis. The actual optimization took place in the Efficient Frontier class. In finance, an efficient frontier is the set of optimal portfolios that offer the highest expected return for a defined level of risk or the lowest risk for a given level of expected return [7]. The set-up of the efficient frontier uses the computed expected returns and covariance matrix, with weights bounded by -2 and 2. In this model, I only consider optimizing to maximize the Sharpe ratio and minimize the volatility. There are two corresponding built-in functions *max_sharpe()* and *min_volatility(),* to achieve the two goals and return the optimized weights on the portfolio. The package contains portfolio performance measurements to calculate the expected return, volatility, and Sharpe ratio for the optimized portfolio. But I tested the resulting weights with the same testing procedure as it is in the previous model.

The results of this model show two significantly diverse patterns of maximizing Sharpe ratio and minimizing volatility methods. From the line chart in Figure 5, I could easily observe that those lines with significant differences between their max and min, i.e., with large volatility, are all based on maximizing Sharpe ratios. Also, when testifying the results of portfolio training

from 2014-2018 training data, I found that it produces abnormally high returns during the year 2020. This is because the stock of Tesla outperformed that year. The influence of this single outperforming stock could also be observed from the exponentially weighted means of the historical data for 2016-2020. The calculated mean of the Tesla stock is significantly larger than all other stocks. I tried to minimize the instability of such stock by extending the dataset, i.e., using 2014-2020 data, but it didn't help. So, I tried to adjust the parameters span and frequency. To discuss in detail, maximizing the Sharpe ratio and minimizing the volatility have different strategies to tune the parameters. A larger time span with a fixed frequency produces better results for the max_sharpe function, while a smaller time span with a fixed frequency creates better outcomes for the min_volatility function. Smaller frequency with fixed span leads to better results for *max_sharpe()*, while changing the frequencies with fixed span turns out to have very subtle changes.

```
AAPL   -0.671947
MSFT   -0.226546
GOOG   -0.178334
AMZN   -0.414067
TSLA    0.370328
FB     -0.500738
BAC    -0.436928
JPM    -0.428485
NVDA   -0.858710
NFLX   -0.391524
Name: 1256, dtype: float64
```

Figure 4: A sample output of exponentially weighted means of 2014-2018 historical returns with frequency equals 252 and span equals 63.

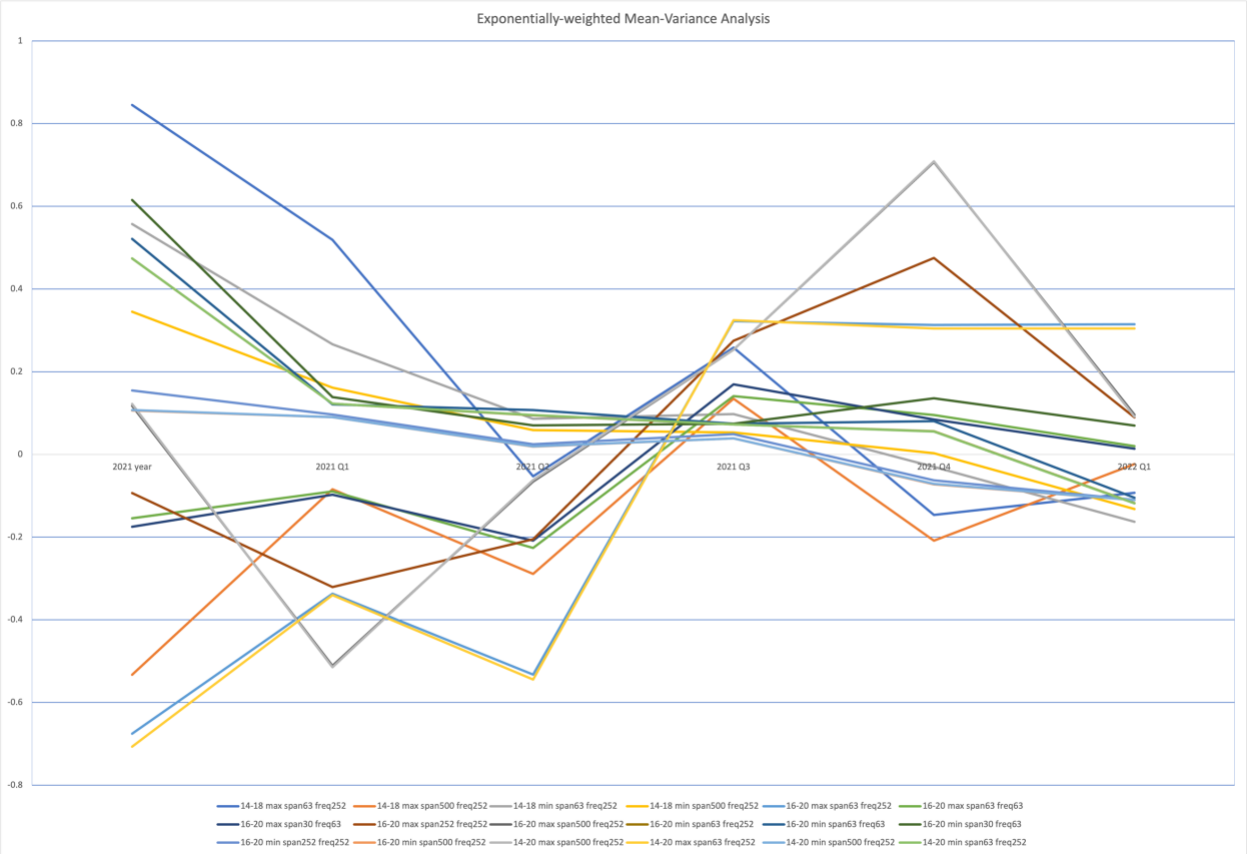


Figure 5: The returns of portfolio with different portfolio weights output from the exponentially weighted model.

The previous two models have advantages that their results are easy to understand and apply, and the parameters are easy to tune. However, the weights generating process is hard to be displayed or personalized. Therefore, I adapted a deep learning method for portfolio optimization [9]. The main structure of my model is a Long Short-Term Memory (LSTM) model. It is a recurrent neural network architecture that can keep track of long-term dependencies of input sequences [8]. The main idea is to predict future prices of stocks and determine how to distribute assets on the stocks. Most operations in this method applied functions from the Tensorflow.keras python library. I directly read in the historical price data from 2014 to current as a data frame. I only kept the dates and adjusted close price, then drop all other columns. I split 80% of historical data in time sequence as training data, around 20% as validation data, and 30 trading days as testing data. Next, I defined a parameter *look_back* which is the number of previous days I retrieved to predict the next day. Then, I used the *TimeseriesGenerator()* to transform the historical price data along with time series parameters. My LSTM model is a 10 LSTM units structure trained along with Adam optimizer and Mean Squared Error loss for 50 epochs. I validate the model with the validation set and generate 30 days "future" predictions for the test set. Figure 6 shows a sample of results generated by this LSTM model. All of the above parameters can be adjusted. I trained 10 models, one for each stock. If time and computation power permits, I could train the model for more epochs and try more different parameters. It is also possible to train one general model for the whole portfolio.

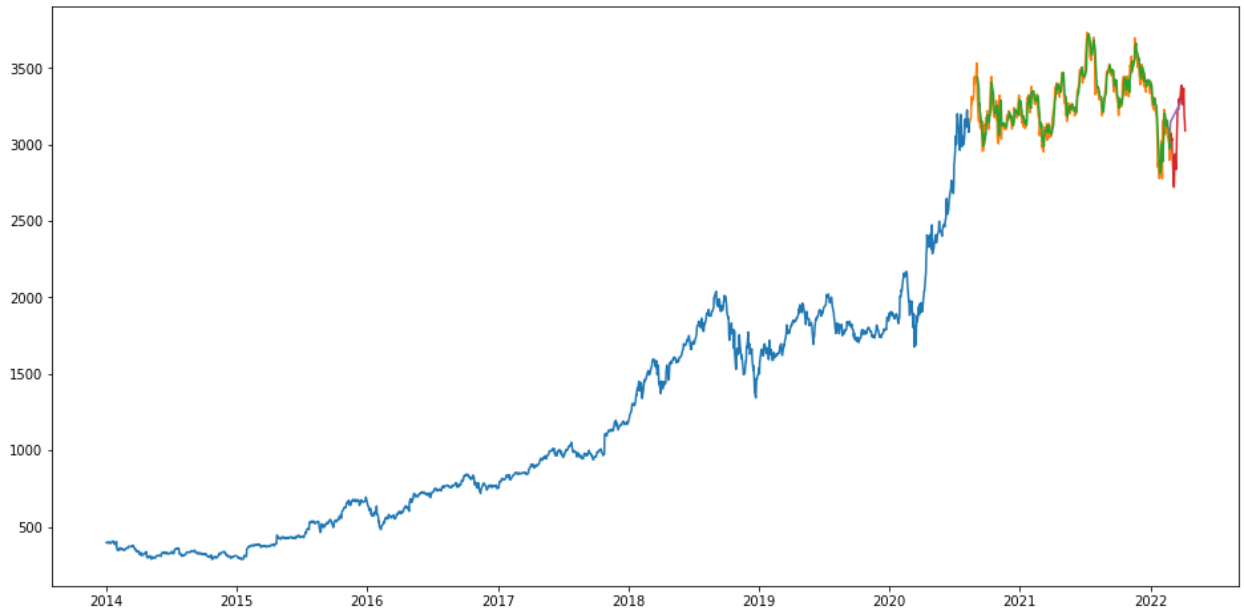


Figure 6: A sample output of the LSTM model with Amazon.com, Inc stock's historical data. The blue line is the training set, the orange line is the ground truth data for validation set, the green line is the validation results generated by the model, the red line is the ground truth data for the 30 days test set, and the purple line is the predicted "future" prices.

The next part of this optimization method is to define how to generate the portfolio weights. I would interpret the predicted results as a trend in the future. For example, I could reasonably guess whether the stock price will go up or down in the future. With these thoughts in

mind, I designed a weight-generating process. I calculated the difference between the last day's price of the prediction period and that of the validation period. Then, I appended the differences into a list and normalized each element by dividing by the sum of these differences. I wanted the weights to sum up to 1, so I divided each element by their sum. This procedure produces a set of reasonable portfolio weights. The test results of this model's output portfolio weights are not yet optimal due to my time and computational power limitations. But the advantage of this model is that it could easily be saved and reloaded. The weights generation process is also more transparent and flexible than other methods.

4. Conclusion

The three different methods implemented and discussed in this project generate various investment recommendations that accomplish different investment goals. The first basic mean-variance analysis produces a comparably stable and average-performing portfolio. The second method applied an exponentially weighted mean to achieve better results. The modified mean-variance optimization method diversifies the effects of different goals so that investors can choose as needed. For example, if investors can tolerate more risks, they might choose to maximize the Sharpe ratio. In fact, this method also allows adding more constraints that may satisfy more investors' requirements. For example, it is possible to let the model minimize the volatility while ensuring a certain level of expected return. For the last deep learning model, it is more flexible and modifiable by the investors. The extendibility of the last model is also incredible. For example, the intermediate output of the LSTM model might be simply used as a future stock price predictor.

However, one thing to note is that these results are only recommendations for investors. None of these three methods generates very promising results. They couldn't guarantee any stable earnings. This is partially due to the extremely dynamic market in recent years. But improvements can absolutely be made. To further optimize these models, more data input is considerably helpful. The first two methods have few parameters to tune, while the LSTM-based model has the most significant potential to be improved. For example, adding more complicated architecture to the LSTM model or optimizing the weights generating process.

References

[1] Lydia Saad and Jeffrey M. Jones. (2021, November 20). What percentage of Americans owns stock? Gallup.com. Retrieved January 26, 2022, from https://news.gallup.com/poll/266807/percentage-americans-owns-stock.aspx

[2] Statista. (2022, January 11). Share of Americans investing money in the stock market 1999–2020. https://www.statista.com/statistics/270034/percentage-of-us-adults-to-have-money-invested-in-the-stock-market/

[3] Mean-Variance Analysis. (2021, October 19). Investopedia. https://www.investopedia.com/terms/m/meanvariance-analysis.asp

[4] Capital Asset Pricing Model (CAPM). (2022, January 6). Investopedia. https://www.investopedia.com/terms/c/capm.asp

[5] *Kenneth R. French - Home Page*. (2022). Dartmouth College. http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/index.html
[6] *User Guide — PyPortfolioOpt 1.5.2 documentation*. (2022). PyPortfolioOpt. https://pyportfolioopt.readthedocs.io/en/latest/UserGuide.html
[7] *What Is the Efficient Frontier?* (2022, February 28). Investopedia. https://www.investopedia.com/terms/e/efficientfrontier.asp#:%7E:text=The%20efficient%20frontier%20is%20the,for%20the%20level%20of%20risk
[8] Wikipedia contributors. (2022, April 14). *Long short-term memory*. Wikipedia. https://en.wikipedia.org/wiki/Long_short-term_memory
[9] Jose, G. V. (2021, December 11). *Predicting Sequential Data using LSTM: An Introduction*. Medium. https://towardsdatascience.com/time-series-forecasting-with-recurrent-neural-networks-74674e289816

Appendix

Check List for Proposal and Final Report
Check that you addressed each item and initial it to the right

1. **Structure of document**: fond, spaces, minimum number of required pages. Font should be 11 or 12. Space should be single or 1.15. Margins should be 1in all around LY
2. All **abbreviations** are defined when first appeared LY
3. All **figures and tables** are labeled with numbers and captions. Figure captions go under the plots and table captions above tables. Make sure you indicate units. LY
4. Figures, tables, diagrams, pictures taken from the web or papers should include the source otherwise it is plagiarism LY
5. No text should be copied and paste from anywhere as it is plagiarism. If you quote someone you should put it under "...." and identify the source as a reference but it should not be more than a couple of lines. LY
6. All listed references should be mentioned in text. Use format [1], [2] LY
7. **Grammar and English** should be carefully checked . No report with typos and grammatical errors will be accepted. Check the tenses. Some students tend to use parts of the proposal that uses the words 'I will' , 'I plan' etc. The final report should state what you did not plan to do. LY

PLIAGIARISM: *Any identified form of plagiarism will be reported to the University and in addition to receiving no credit for EE590 the graduation of the student may be affected. All past reports are kept and a software program is used to identify overlaps with past reports and literature. If you use figures or tables or diagrams from a website or papers you need to indicate that and include the reference..*

I confirm that my proposal/report abides with the above guidelines and no material was copied from any source that is not indicated in the report and referenced accordingly.

Name Linyin Yang

Signature 

Basic Mean-Variance Analysis

First, we need to estimate mean and variance of the excess market returns, simply by taking average

E[RMRF] =	0.0331265		SD(RMRF) =	0.85256106					

We also need to estimate a risk-free rate, here I take the most recent risk-free rate

RF =	0.0022267

Compute the alphas of all stocks using the intercept function

AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
-0.0018529	-0.0016329	-0.0020077	-0.00142183	-0.00165854	-0.0017661	-0.0021461	-0.0020235	-0.0006776	-0.0010548

Compute the betas of all stocks by using the SLOPE function

AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
0.0109261	0.0123231	0.011659	0.013332336	0.01386033	0.0124701	0.0133063	0.0117901	0.0151963	0.0150685

Compute standard deviation of the epsilon of each stock

AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
0.0122694	0.0105278	0.0113347	0.016160167	0.02576368	0.0159894	0.0117008	0.0089942	0.0222004	0.0241018

Compute Expected excess returns (set alpha to zero according to CAPM holds condition)

AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
0.0003619	0.0004082	0.0003862	0.000441654	0.00045914	0.0004131	0.0004408	0.0003906	0.0005034	0.0004992

Compute Expected returns

AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
0.0025887	0.002635	0.002613	0.002668384	0.00268587	0.0026398	0.0026675	0.0026173	0.0027301	0.0027259

Compute variance of each stock by beta^2*Var(RM)+Var(epsilon)

AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
0.0002373	0.0002212	0.0002273	0.000390351	0.0008034	0.0003687	0.0002656	0.0001819	0.0006607	0.0007459

So, SD of each stock

AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
0.0154049	0.0148733	0.0150758	0.01975731	0.02834438	0.0192013	0.0162974	0.0134883	0.0257043	0.0273119

Variance-Covariance Matrix

		AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
		0.010926077	0.01232308	0.011659	0.0133323	0.0138603	0.0124701	0.0133063	0.0117901	0.0151963	0.0150685
AAPL	0.0109261	0.00023731	9.7867E-05	9.259E-05	0.0001059	0.0001101	9.903E-05	0.0001057	9.363E-05	0.0001207	0.0001197
MSFT	0.0123231	9.78666E-05	0.00022121	0.0001044	0.0001194	0.0001241	0.0001117	0.0001192	0.0001056	0.0001361	0.000135
GOOG	0.011659	9.25926E-05	0.00010443	0.0002273	0.000113	0.0001175	0.0001057	0.0001128	9.992E-05	0.0001288	0.0001277
AMZN	0.0133323	0.000105882	0.00011942	0.000113	0.0003904	0.0001343	0.0001208	0.0001289	0.0001143	0.0001473	0.000146
TSLA	0.0138603	0.000110075	0.00012415	0.0001175	0.0001343	0.0008034	0.0001256	0.0001341	0.0001188	0.0001531	0.0001518
FB	0.0124701	9.90346E-05	0.0001117	0.0001057	0.0001208	0.0001256	0.0003687	0.0001206	0.0001069	0.0001377	0.0001366
BAC	0.0133063	0.000105675	0.00011919	0.0001128	0.0001289	0.0001341	0.0001206	0.0002656	0.000114	0.000147	0.0001457
JPM	0.0117901	9.36342E-05	0.00010561	9.992E-05	0.0001143	0.0001188	0.0001069	0.000114	0.0001819	0.0001302	0.0001291
NVDA	0.0151963	0.000120685	0.00013612	0.0001288	0.0001473	0.0001531	0.0001377	0.000147	0.0001302	0.0006607	0.0001664
NFLX	0.0150685	0.00011967	0.00013497	0.0001277	0.000146	0.0001518	0.0001366	0.0001457	0.0001291	0.0001664	0.0007459

Correlation Matrix

			AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
		betas	0.015404885	0.0148733	0.0150758	0.0197573	0.0283444	0.0192013	0.0162974	0.0134883	0.0257043	0.0273119
	betas											
AAPL	0.0154049		1	0.42713877	0.3986924	0.3478846	0.2520946	0.3348088	0.420917	0.450628	0.3047812	0.2844307
MSFT	0.0148733		0.427138768	1	0.4657409	0.4063887	0.2944896	0.391114	0.491703	0.5264105	0.3560365	0.3322637
GOOG	0.0150758		0.398692427	0.46574091	1	0.3793243	0.2748774	0.3650668	0.4589568	0.4913529	0.3323254	0.3101358
AMZN	0.0197573		0.347884624	0.40638871	0.3793243	1	0.2398481	0.3185441	0.4004691	0.4287368	0.2899751	0.2706133
TSLA	0.0283444		0.252094615	0.29448961	0.2748774	0.2398481	1	0.230833	0.2902	0.3106842	0.2101305	0.1961
FB	0.0192013		0.334808846	0.39111397	0.3650668	0.3185441	0.230833	1	0.3854169	0.4126221	0.279076	0.2604419
BAC	0.0162974		0.420916953	0.49170296	0.4589568	0.4004691	0.2902	0.3854169	1	0.5187427	0.3508504	0.3274239
JPM	0.0134883		0.450627984	0.52641052	0.4913529	0.4287368	0.3106842	0.4126221	0.5187427	1	0.3756157	0.3505356
NVDA	0.0257043		0.304781173	0.35603651	0.3323254	0.2899751	0.2101305	0.279076	0.3508504	0.3756157	1	0.2370839
NFLX	0.0273119		0.28443075	0.33226374	0.3101358	0.2706133	0.1961	0.2604419	0.3274239	0.3505356	0.2370839	1

Portfolio Weights

	AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
	0.10444249	0.15999291	0.13058658	0.073463468	0.03004771	0.07018824	0.13985776	0.20972603	0.04436758	0.03732723

Stastical Input

	AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
Means	0.00258867	0.00263495	0.00261295	0.002668384	0.00268587	0.00263982	0.00266752	0.0026173	0.00273013	0.0027259
SDs	0.01540489	0.0148733	0.01507578	0.01975731	0.02834438	0.01920133	0.01629736	0.01348832	0.02570429	0.02731186

	AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
AAPL	1	0.42713877	0.39869243	0.347884624	0.25209462	0.33480885	0.42091695	0.45062798	0.30478117	0.28443075
MSFT	0.42713877	1	0.46574091	0.406388712	0.29448961	0.39111397	0.49170296	0.52641052	0.35603651	0.33226374
GOOG	0.39869243	0.46574091	1	0.379324272	0.27487736	0.36506679	0.45895681	0.49135294	0.3323254	0.31013583
AMZN	0.34788462	0.40638871	0.37932427	1	0.23984806	0.3185441	0.40046915	0.42873684	0.28997515	0.27061333
TSLA	0.25209462	0.29448961	0.27487736	0.239848063	1	0.230833	0.29019999	0.31068418	0.21013051	0.19609997
FB	0.33480885	0.39111397	0.36506679	0.318544103	0.230833	1	0.3854169	0.41262211	0.27907599	0.26044192
BAC	0.42091695	0.49170296	0.45895681	0.400469148	0.29019999	0.3854169	1	0.51874269	0.35085039	0.3274239
JPM	0.45062798	0.52641052	0.49135294	0.428736841	0.31068418	0.41262211	0.51874269	1	0.37561567	0.35053559
NVDA	0.30478117	0.35603651	0.3323254	0.28997515	0.21013051	0.27907599	0.35085039	0.37561567	1	0.23708392
NFLX	0.28443075	0.33226374	0.31013583	0.270613334	0.19609997	0.26044192	0.3274239	0.35053559	0.23708392	1

	Weights	SDs						Portfolio Variance				
AAPL	0.10444249	0.01540489	2.58864E-06	1.6354E-06	1.2629E-06	8.124E-07	3.4544E-07	7.2599E-07	1.5436E-06	2.05099E-06	5.59236E-07	4.6654E-07
MSFT	0.15999291	0.0148733	1.63536E-06	5.6626E-06	2.1819E-06	1.4036E-06	5.9684E-07	1.2543E-06	2.6669E-06	3.54358E-06	9.66215E-07	8.06061E-07
GOOG	0.13058658	0.01507578	1.26285E-06	2.1819E-06	3.8758E-06	1.0839E-06	4.6089E-07	9.686E-07	2.0595E-06	2.73641E-06	7.46128E-07	6.22454E-07
AMZN	0.07346347	0.01975731	8.124E-07	1.4036E-06	1.0839E-06	2.1067E-06	2.9649E-07	6.2311E-07	1.3249E-06	1.76035E-06	4.79989E-07	4.00429E-07
TSLA	0.03004771	0.02834438	3.45444E-07	5.9684E-07	4.6089E-07	2.9649E-07	7.2537E-07	2.6495E-07	5.6335E-07	7.48527E-07	2.04098E-07	1.70268E-07
FB	0.07018824	0.01920133	7.25986E-07	1.2543E-06	9.686E-07	6.2311E-07	2.6495E-07	1.8163E-06	1.1839E-06	1.57311E-06	4.28933E-07	3.57836E-07
BAC	0.13985776	0.01629736	1.5436E-06	2.6669E-06	2.0595E-06	1.3249E-06	5.6335E-07	1.1839E-06	5.1953E-06	3.34477E-06	9.12005E-07	7.60836E-07
JPM	0.20972603	0.01348832	2.05099E-06	3.5436E-06	2.7364E-06	1.7604E-06	7.4853E-07	1.5731E-06	3.3448E-06	8.0024E-06	1.21178E-06	1.01093E-06
NVDA	0.04436758	0.02570429	5.59236E-07	9.6622E-07	7.4613E-07	4.7999E-07	2.041E-07	4.2893E-07	9.12E-07	1.21178E-06	1.3006E-06	2.75645E-07
NFLX	0.03732723	0.02731186	4.6654E-07	8.0606E-07	6.2245E-07	4.0043E-07	1.7027E-07	3.5784E-07	7.6084E-07	1.01093E-06	2.75645E-07	1.03933E-06

Potfolio Variance	0.0001311
Portfolio SD	0.0114501
Portfolio Mean	0.00264004
risk-free rate	0.00222673
Sharpe ratio	0.03609687

Testing scripts

In [132]:
```
import csv
import glob
```

In [133]:
```
# initially we have 1,000,000 dollars
initial = 1000000
```

In [193]:
```
# portfolio weights
# dict{stock_name: weights}
weights = {}
weights['AAPL']=0.03515294
weights['MSFT']=0.19514679
weights['GOOG']=1.1467935
weights['AMZN']=1.39732297
weights['TSLA']=-0.63920717
weights['FB']=0.33857165
weights['BAC']=-0.13222546
weights['JPM']=-0.06169491
weights['NVDA']=-0.28901159
weights['NFLX']=0.00915127
```

In [198]:
```
# time period
start = '2021-01-04'
end = '2021-12-31'
```

In [199]:
```
# pull up prices from each csv file
files=glob.glob('./portfolio/*')
start_price = {}
end_price = {}
stocks = []
for file in files:
    stock_name = file.split('/')[2][:-4]
    stocks.append(stock_name)
    with open(file) as f:
        reader=csv.reader(f)
        for row in reader:
            #print(row)
            if row[0] == start:
                start_price[stock_name] = row[5]
            elif row[0] == end:
                end_price[stock_name] = row[5]
```

In [200]:
```
# calculate returns on each stock and sum up for results
# initially
returns = 0
for s in stocks:
    returns += initial*weights[s]/float(start_price[s])*float(end_price[s])
returns = (returns-initial)/initial
```

In [201]:
```
returns
```

Out[201]:
```
0.1575354095740621
```

Exponentially weighted mean Mean-Variance Analysis

Exponentially_weighted

April 20, 2022

[26]:
```
from pypfopt import risk_models
from pypfopt import expected_returns
from pypfopt.expected_returns import ema_historical_return
from pypfopt.risk_models import exp_cov
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt.plotting import plot_efficient_frontier
from pypfopt.plotting import plot_weights
from pypfopt.cla import CLA
```

[27]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import numpy as np
```

[28]:
```
path='./training_mv.xlsx'
```

[29]:
```
train = pd.read_excel(path, sheet_name="16_returns", usecols = "B:K")
```

[39]:
```
train
```

[39]:

	AAPL	MSFT	GOOG	AMZN	TSLA	FB	BAC	JPM	NVDA	NFLX
0	-0.025059	0.004562	0.000998	-0.005024	0.000090	0.004989	0.000000	0.001729	0.016064	-0.020917
1	-0.019570	-0.018165	0.001400	-0.001799	-0.019648	0.002336	-0.021302	-0.014436	-0.041350	0.093071
2	-0.042204	-0.034783	-0.023170	-0.039058	-0.015477	-0.049043	-0.036070			
3	0.005288	0.003067	-0.016410	-0.001464	-0.021563	-0.006025	-0.019355			
4	0.016192	-0.000574	0.002184	0.017610	-0.014929	0.001849	0.007237			
…	…	…	…	…	…	…	…			
1253	0.007712	0.007827	0.003735	-0.003949	0.024444	-0.002648	-0.002995			
1254	0.035766	0.009921	0.021416	0.035071	0.002901	0.035901	0.005674			
1255	-0.013315	-0.003601	-0.009780	0.011584	0.003465	-0.000794	-0.003983			
1256	-0.008527	-0.011019	-0.010917	-0.010882	0.043229	-0.017740	-0.001000			
1257	-0.007703	0.003338	0.007105	-0.008801	0.015674	0.004745	0.011007			

```
2    -0.040439 -0.039645 -0.026513
3    -0.022399 -0.021466 -0.027671
4    -0.001527  0.001688  0.032139
…          …         …         …
1253 -0.004398 -0.001192 -0.000991
1254  0.006585 -0.007215  0.010020
1255 -0.002633  0.003353  0.022634
1256  0.002800  0.015645 -0.011830
1257  0.013641 -0.006903  0.030767

[1258 rows x 10 columns]
```

[59]:
```
mu = expected_returns.ema_historical_return(train, frequency=252, returns_data␣
↪= True, span=500)
sigma = risk_models.exp_cov(train, frequency=252, returns_data = True, span=500)
```

[60]:
```
mu
```

[60]:
```
AAPL    0.905012
MSFT    0.484257
GOOG    0.361818
AMZN    0.530702
TSLA    4.998805
FB      0.385916
BAC     0.287654
JPM     0.340006
NVDA    0.994242
NFLX    0.536583
Name: 1257, dtype: float64
```

[61]:
```
rf = pd.read_excel(path, sheet_name="16_returns", usecols = "L")
```

[62]:
```
risk_free_rate = rf['RF'][-365:].mean()
```

[63]:
```
risk_free_rate
```

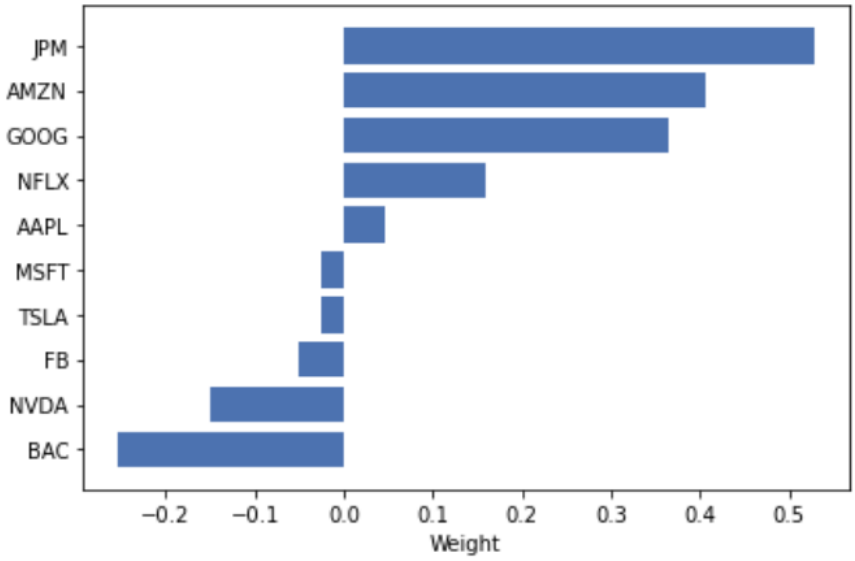
[63]:
```
0.0032547945205479473
```

[67]:
```
ef = EfficientFrontier(mu, sigma, weight_bounds=(-2,2))
raw_weights_exp = ef.min_volatility()
#raw_weights_exp = ef.max_sharpe(risk_free_rate=risk_free_rate)

plot_weights(raw_weights_exp)
ef.portfolio_performance(verbose = True, risk_free_rate =risk_free_rate)
```

```
Expected annual return: 27.0%
Annual volatility: 27.0%
```

```
Sharpe Ratio: 0.99
```

[67]:
```
(0.2695051026220518, 0.26994868624841256, 0.9862996994047032)
```



[68]:
```
raw_weights_exp.values()
```

[68]:
```
odict_values([0.0471837154487892, -0.0246266022463904, 0.3653663875908934,
0.4063420628333646, -0.0264238095260919, -0.0503740717440858,
-0.2534360867095144, 0.5275825967027327, -0.1498171754484958,
0.1582029830987984])
```

[69]:
```
raw_weights_exp
```

[69]:
```
OrderedDict([('AAPL', 0.0471837154487892),
             ('MSFT', -0.0246266022463904),
             ('GOOG', 0.3653663875908934),
             ('AMZN', 0.4063420628333646),
             ('TSLA', -0.0264238095260919),
             ('FB', -0.0503740717440858),
             ('BAC', -0.2534360867095144),
             ('JPM', 0.5275825967027327),
             ('NVDA', -0.1498171754484958),
             ('NFLX', 0.1582029830987984)])
```

LSTM model

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
     from matplotlib.pylab import rcParams
     rcParams['figure.figsize']=20,10
     from keras.models import Sequential
     from keras.layers import LSTM,Dropout,Dense
     from sklearn.preprocessing import MinMaxScaler
     import keras
     import tensorflow as tf
     from keras.preprocessing.sequence import TimeseriesGenerator
     from keras.models import load_model
     import glob
```

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

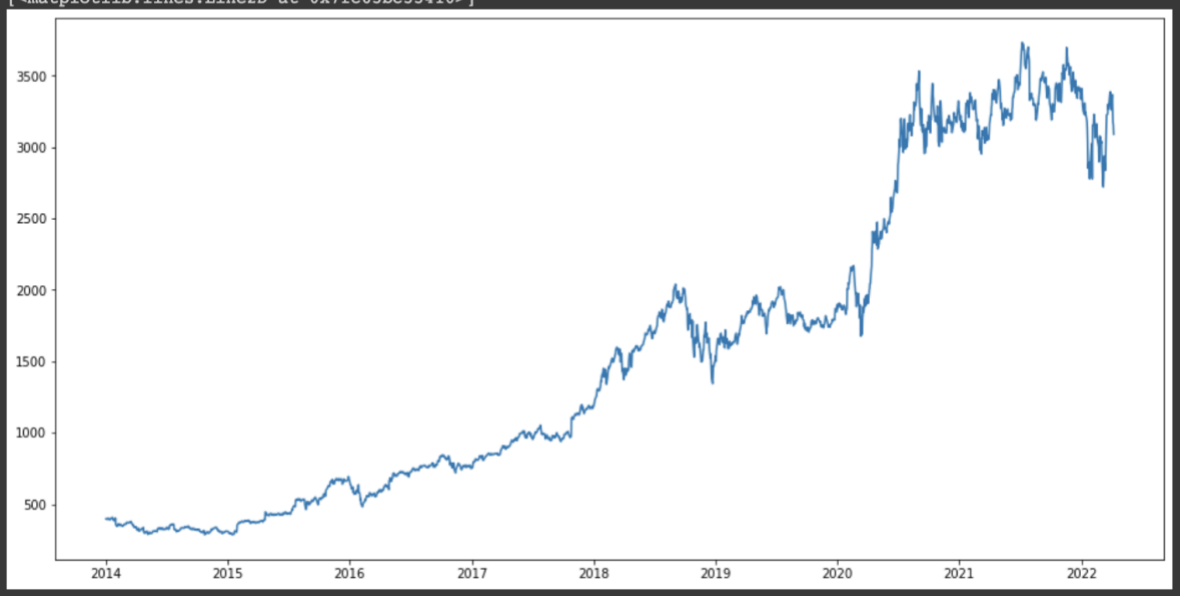
```
[ ]  ud=[0]*10
     change=[0]*10
```

```
[ ]  i=3
     path = '/content/drive/MyDrive/portfolio/'
     stock = glob.glob(path+'*.csv')
     df=pd.read_csv(stock[i])
     name = stock[i].split('/')[-1][:-4]
```

```
     df["Date"]=pd.to_datetime(df.Date,format="%Y-%m-%d")
     df.set_axis(df['Date'], inplace=True)
     df.drop(columns=['Open', 'High', 'Low', 'Volume'], inplace=True)
     plt.figure(figsize=(16,8))
     plt.plot(df["Adj Close"],label='Adj. Close Price history')
```

```
df["Date"]=pd.to_datetime(df.Date,format="%Y-%m-%d")
df.set_axis(df['Date'], inplace=True)
df.drop(columns=['Open', 'High', 'Low', 'Volume'], inplace=True)
plt.figure(figsize=(16,8))
plt.plot(df["Adj Close"],label='Adj. Close Price history')
```

```
[<matplotlib.lines.Line2D at 0x7fc63bc33410>]
```




```
close_data = df['Adj Close'].values
close_data = close_data.reshape((-1,1))

split_percent = 0.80
split = int(split_percent*len(close_data))

close_train = close_data[:split]
close_valid = close_data[split:len(close_data)-30]
close_test = close_data[len(close_data)-30:]

date_train = df['Date'][:split]
date_valid = df['Date'][split:len(close_data)-30]
date_test = df['Date'][len(close_data)-30:]

print(len(close_train))
print(len(close_valid))
print(len(close_test))
```

```
1666
387
30
```

```
look_back = 15
train_generator = TimeseriesGenerator(close_train, close_train, length=look_back, batch_size=20)
valid_generator = TimeseriesGenerator(close_valid, close_valid, length=look_back, batch_size=1)
```

```
[ ] model = load_model(path+name+'.h5')
```

```
model = Sequential()
model.add(
    LSTM(10,
        activation='relu',
        input_shape=(look_back,1))
)
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

num_epochs = 25
model.fit_generator(train_generator, epochs=num_epochs, verbose=1)
```

```
Epoch 1/25
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: UserWarning: `Model.fit_gen
  # This is added back by InteractiveShellApp.init_path()
83/83 [==============================] - 2s 7ms/step - loss: 835.7691
Epoch 2/25
83/83 [==============================] - 1s 7ms/step - loss: 16.1469
Epoch 3/25
83/83 [==============================] - 1s 7ms/step - loss: 17.0804
Epoch 4/25
83/83 [==============================] - 1s 7ms/step - loss: 15.5811
Epoch 5/25
83/83 [==============================] - 1s 7ms/step - loss: 16.5967
Epoch 6/25
83/83 [==============================] - 1s 7ms/step - loss: 15.1493
Epoch 7/25
83/83 [==============================] - 1s 7ms/step - loss: 15.8041
Epoch 8/25
83/83 [==============================] - 1s 7ms/step - loss: 15.5220
Epoch 9/25
83/83 [==============================] - 1s 7ms/step - loss: 16.1709
Epoch 10/25
83/83 [==============================] - 1s 7ms/step - loss: 16.0000
Epoch 11/25
83/83 [==============================] - 1s 7ms/step - loss: 15.5038
Epoch 12/25
83/83 [==============================] - 1s 7ms/step - loss: 16.3874
```

```
[ ] model.save(path+name+'.h5')
```

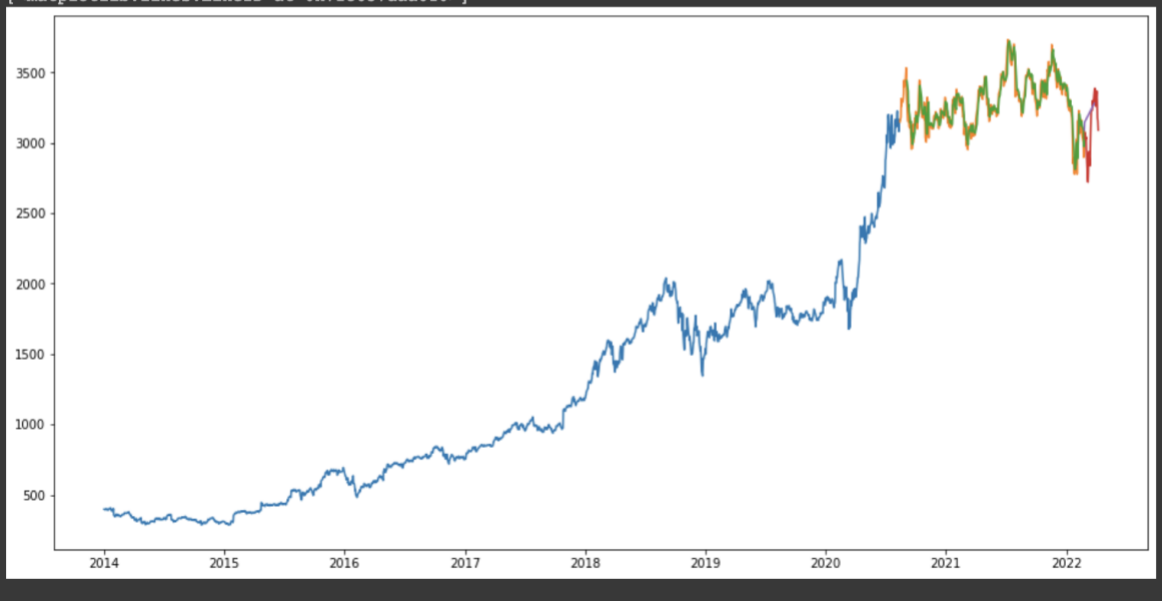
```
[ ] validation = model.predict_generator(valid_generator)

    close_train_reshape = close_train.reshape((-1))
    close_valid_reshape = close_valid.reshape((-1))
    close_test_reshape = close_test.reshape((-1))
    validation_reshape = validation.reshape((-1))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `Model.pred
  """Entry point for launching an IPython kernel.
```

```
close_data_reshape = close_data.reshape((-1))

def predict(num_prediction, model):
    prediction_list = close_data_reshape[-look_back:]

    for _ in range(num_prediction):
        x = prediction_list[-look_back:]
        x = x.reshape((1, look_back, 1))
        out = model.predict(x)[0][0]
        prediction_list = np.append(prediction_list, out)
    prediction_list = prediction_list[look_back-1:]

    return prediction_list

def predict_dates(num_prediction):
    last_date = df['Date'].values[-num_prediction-1]
    prediction_dates = pd.date_range(last_date, periods=num_prediction+1).tolist()
    return prediction_dates

num_prediction = 30
forecast = predict(num_prediction, model)
forecast_dates = predict_dates(num_prediction)
```

```
plt.rcParams["figure.figsize"] = (16,8)
plt.plot(date_train, close_train_reshape)
plt.plot(date_valid, close_valid_reshape)
plt.plot(date_valid[look_back:], validation_reshape)
#plt.plot(date_valid[:(len(date_valid)-look_back)], validation_reshape)
plt.plot(date_test, close_test_reshape)
plt.plot(forecast_dates, forecast)
```

```
[<matplotlib.lines.Line2D at 0x7fc637daa610>]
```



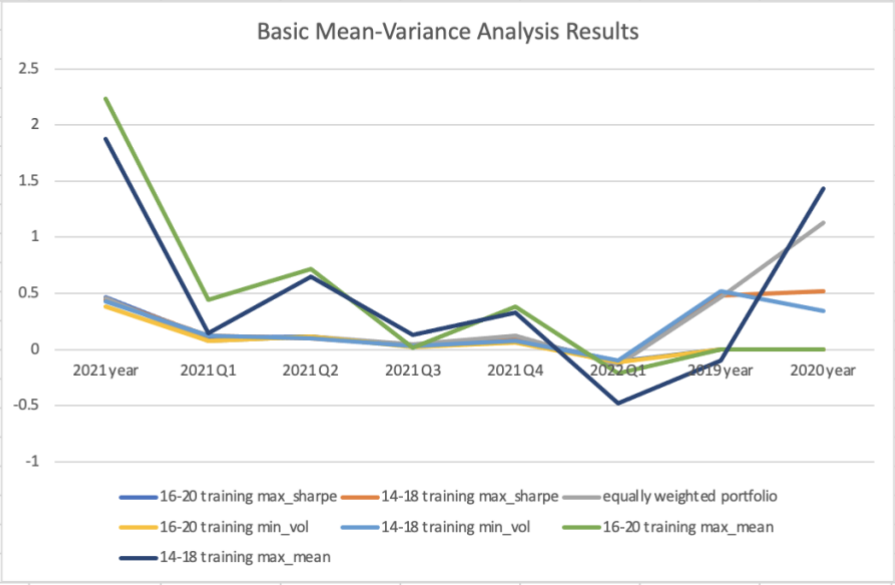

```
[ ] weights={}
    for i in range(10):
        n = stock[i].split('/')[-1][:-4]
        weights[n]=stand_change[i]
```

```
weights
```

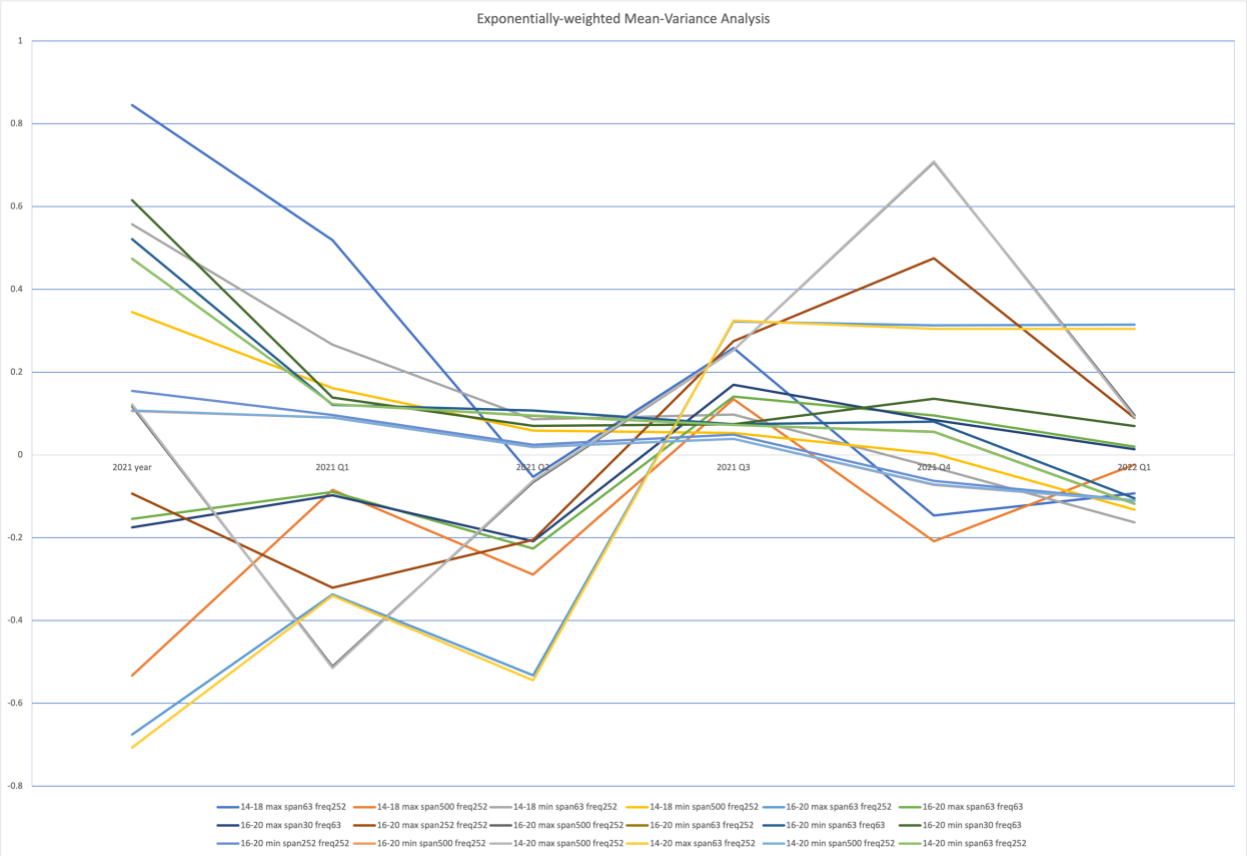
```
{'AAPL': 0.017576467845318636,
 'AMZN': 0.6986614849025095,
 'BAC': -0.0661127299352913,
 'FB': 0.16928582720578217,
 'GOOG': 0.5733967492107045,
 'JPM': -0.03084745261822503,
 'MSFT': 0.0975733963564137,
 'NFLX': 0.004575636591458588,
 'NVDA': -0.14450579532644853,
 'TSLA': -0.31960358423222207}
```

Results

A	B	C	D	E	F	G	H	I
Column1	2021 year	2021 Q1	2021 Q2	2021 Q3	2021 Q4	2022 Q1	2019 year	2020 year
16-20 training max_sharpe	0.46913628	0.1112053	0.1147438	0.0343955	0.1008581	-0.108859	-	-
14-18 training max_sharpe	0.44451388	0.1233929	0.1023062	0.0354603	0.0841375	-0.116033	0.4796614	0.51703346
equally weighted portfolio	0.45545516	0.0738052	0.1181952	0.0429705	0.1223157	-0.136424	0.4644331	1.13079908
16-20 training min_vol	0.37965832	0.0795214	0.1168231	0.0248075	0.0622613	-0.114321	-	-
14-18 training min_vol	0.4271919	0.1248559	0.0965409	0.0349645	0.078379	-0.101219	0.52046159	0.34019442
16-20 training max_mean	2.23834494	0.4406638	0.7147924	0.0126573	0.3807127	-0.212104	-	-
14-18 training max_mean	1.88050666	0.1435911	0.6455541	0.1294583	0.3255449	-0.481997	-0.0977683	1.43520411



Column1	2021 year	2021 Q1	2021 Q2	2021 Q3	2021 Q4	2022 Q1	2019 year	2020 year
14-18 max span63 freq252	0.84518616	0.5186668	-0.05282	0.2582764	-0.146288	-0.092899	-0.73653139	4.37753829
14-18 max span500 freq252	-0.53273837	-0.084331	-0.28886	0.1347837	-0.208394	-0.023237	-0.31064959	1.93261403
14-18 min span63 freq252	0.55696932	0.2661888	0.0860071	0.0974256	-0.030251	-0.16292	0.50253423	-0.08676976
14-18 min span500 freq252	0.34532768	0.161587	0.0589715	0.0528911	0.0030907	-0.132361	0.53099558	0.1962286
16-20 max span63 freq252	-0.67509715	-0.336845	-0.532357	0.3219641	0.3127434	0.3149316	-	-
16-20 max span63 freq63	-0.15447967	-0.089579	-0.226124	0.1410364	0.0950755	0.0199107	-	-
16-20 max span30 freq63	-0.17482278	-0.097173	-0.208607	0.1693928	0.08468	0.0139585	-	-
16-20 max span252 freq252	-0.09330091	-0.320846	-0.205116	0.2748159	0.4751846	0.0898012	-	-
16-20 max span500 freq252	0.11732844	-0.511236	-0.065646	0.2530202	0.7067889	0.0957596	-	-
16-20 min span63 freq252	0.47395542	0.1225642	0.0949276	0.0726993	0.055907	-0.117944	-	-
16-20 min span63 freq63	0.52137863	0.1208405	0.1071293	0.0744047	0.0802274	-0.104537	-	-
16-20 min span30 freq63	0.61524584	0.138851	0.0702768	0.0744047	0.1356181	0.0696406	-	-
16-20 min span252 freq252	0.15447187	0.0962782	0.0247287	0.049053	-0.062453	-0.10962	-	-
16-20 min span500 freq252	0.10609056	0.0901968	0.0190241	0.0388026	-0.072399	-0.110259	-	-
14-20 max span500 freq252	0.12182885	-0.514426	-0.062278	0.2529424	0.7088627	0.091184	-	-
14-20 max span63 freq252	-0.70642375	-0.340403	-0.544164	0.3245863	0.3045873	0.3045873	-	-
14-20 min span500 freq252	0.1075543	0.0900626	0.0195571	0.0386913	-0.071425	-0.110596	-	-
14-20 min span63 freq252	0.47382541	0.1228372	0.0950793	0.0725558	0.0553759	-0.118593	-	-



Testing Raw Results – Basic
16-20 training
max_sharpe basic_mv
weights['AAPL']=0.125124264
weights['MSFT']=0.211227244
weights['GOOG']=0.151807157
weights['AMZN']=0.069933648
weights['TSLA']=0.02207931
weights['FB']=0.071933376
weights['BAC']=0.119437172
weights['JPM']=0.14254742
weights['NVDA']=0.05021227
weights['NFLX']=0.035698139
21 1yr_test 0104-1231
return = 0.46913627956245185
21 1q_test 0104-0331
return = 0.11120528217450157
21 2q_test 0401-0630
return = 0.11474383300828189

21 3q_test 0701-0930
return = 0.0343954612177423
21 4q_test 1001-1231
return = 0.10085811134663107
22 1q_test 0103-0331
return = -0.10885856071664579

14-18 training
max_sharpe basic_mv
weights['AAPL']=0.104442993
weights['MSFT']=0.159992833
weights['GOOG']=0.130586685
weights['AMZN']=0.073463446
weights['TSLA']=0.030047628
weights['FB']=0.07018812
weights['BAC']=0.139857611
weights['JPM']=0.209726202
weights['NVDA']=0.044367225
weights['NFLX']=0.037327258
21 1yr_test 0104-1231
return = 0.44451388436805156
21 1q_test 0104-0331
return = 0.12339290603638836
21 2q_test 0401-0630
return = 0.10230618353956915
21 3q_test 0701-0930
return = 0.03546026558081387
21 4q_test 1001-1231
return = 0.08413746506465879
22 1q_test 0103-0331
return = -0.11603346348006419
19 1yr_test 0102-1231
return = 0.47966140164810045
20 1yr_test 0102-1231
return = 0.5170334595351345

equally weighted portfolio
weights['AAPL']=0.1
weights['MSFT']=0.1
weights['GOOG']=0.1

weights['AMZN']=0.1
weights['TSLA']=0.1
weights['FB']=0.1
weights['BAC']=0.1
weights['JPM']=0.1
weights['NVDA']=0.1
weights['NFLX']=0.1
21 1yr_test 0104-1231
return = 0.45545516180020057
21 1q_test 0104-0331
return = 0.07380522166148573
21 2q_test 0401-0630
return = 0.11819516697210795
21 3q_test 0701-0930
return = 0.04297053238642355
21 4q_test 1001-1231
return = 0.1223157094821625
22 1q_test 0103-0331
return = -0.13642376791890443
19 1yr_test 0102-1231
return = 0.46443310128954846
20 1yr_test 0102-1231
return = 1.130799082606447

14-18 training
max_mean basic_mv
weights['AAPL']=-2
weights['MSFT']=0
weights['GOOG']=0
weights['AMZN']=0
weights['TSLA']=0
weights['FB']=0
weights['BAC']=0
weights['JPM']=0
weights['NVDA']=2
weights['NFLX']=1
21 1yr_test 0104-1231
return = 1.8805066586778858
21 1q_test 0104-0331
return = 0.14359114015100546

21 2q_test 0401-0630
return = 0.6455541341184085
21 3q_test 0701-0930
return = 0.12945833753547142
21 4q_test 1001-1231
return = 0.3255448756905105
22 1q_test 0103-0331
return = -0.4819972734439343
19 1yr_test 0102-1231
return = -0.09776833347322979
20 1yr_test 0102-1231
return = 1.435204107269241

16-20 training
max_mean basic_mv
weights['AAPL']=-2
weights['MSFT']=0
weights['GOOG']=0
weights['AMZN']=0
weights['TSLA']=0
weights['FB']=0
weights['BAC']=1
weights['JPM']=0
weights['NVDA']=2
weights['NFLX']=0
21 1yr_test 0104-1231
return = 2.238344944402009
21 1q_test 0104-0331
return = 0.4406637764064951
21 2q_test 0401-0630
return = 0.7147924104135925
21 3q_test 0701-0930
return = 0.012657276879719691
21 4q_test 1001-1231
return = 0.3807127022050149
22 1q_test 0103-0331
return = -0.21210412014361868

14-18 training

min_vol basic_mv
weights['AAPL']=0.199329184
weights['MSFT']=0.161732422
weights['GOOG']=0.184222786
weights['AMZN']=0.035205367
weights['TSLA']=0.006977435
weights['FB']=0.065141498
weights['BAC']=0.068829455
weights['JPM']=0.278561852
weights['NVDA']=0
weights['NFLX']=0
21 1yr_test 0104-1231
return = 0.4271919006044972
21 1q_test 0104-0331
return = 0.12485590238213819
21 2q_test 0401-0630
return = 0.09654090536529035
21 3q_test 0701-0930
return = 0.034964457516379076
21 4q_test 1001-1231
return = 0.07837896090981085
22 1q_test 0103-0331
return = -0.10121874695374362
19 1yr_test 0102-1231
return = 0.5204615917373169
20 1yr_test 0102-1231
return = 0.34019441605402856

16-20 training
min_vol basic_mv
weights['AAPL']=0.101186347
weights['MSFT']=0.133085706
weights['GOOG']=0.297234733
weights['AMZN']=0.217809721
weights['TSLA']=0
weights['FB']=0.119045342
weights['BAC']=0
weights['JPM']=0.047308705
weights['NVDA']=0
weights['NFLX']=0.084329446

21 1yr_test 0104-1231
return = 0.37965831797759375
21 1q_test 0104-0331
return = 0.07952142497536563
21 2q_test 0401-0630
return = 0.11682311952861119
21 3q_test 0701-0930
return = 0.024807514630616527
21 4q_test 1001-1231
return = 0.06226127510974277
22 1q_test 0103-0331
return = -0.11432072832813021

Testing Raw Results – Exp.
14-18 training
ema max_sharpe freq=252 span=63 rf= 0.006161643835616424
weights['AAPL']=-2.0000000000000004
weights['MSFT']=0.6544619033821895
weights['GOOG']=2.0
weights['AMZN']=-0.6230087545366915
weights['TSLA']=0.7613681199557851
weights['FB']=-0.0579818239049335
weights['BAC']=-0.0916599559005109
weights['JPM']=0.4607398565456726
weights['NVDA']=-0.5079596792052943
weights['NFLX']=0.4040403336637828
21 1yr_test 0104-1231
return = 0.845186160514879
21 1q_test 0104-0331
return = 0.5186667778287369
21 2q_test 0401-0630
return = -0.05281999769422086
21 3q_test 0701-0930
return = 0.25827638137188996
21 4q_test 1001-1231
return = -0.14628787127937748
22 1q_test 0103-0331
return = -0.09289869367541
19 1yr_test 0102-1231
return = -0.7365313900451814

20 1yr_test 0102-1231
return = 4.377538290179105

14-18 training
ema max_sharpe freq=252 span=500 rf= 0.006161643835616424
weights['AAPL']=-1.135539467781376
weights['MSFT']=2.0
weights['GOOG']=-0.4059415984062792
weights['AMZN']=1.1938902114929504
weights['TSLA']=0.2593882173612716
weights['FB']=-1.0257227319178985
weights['BAC']=-1.730031939822053
weights['JPM']=1.9999999999999996
weights['NVDA']=-0.4853065057258307
weights['NFLX']=0.3292638147992158
21 1yr_test 0104-1231
return = -0.5327383696011193
21 1q_test 0104-0331
return = -0.08433095653760596
21 2q_test 0401-0630
return = -0.28885951109952945
21 3q_test 0701-0930
return = 0.1347837376666495
21 4q_test 1001-1231
return = -0.20839414504329046
22 1q_test 0103-0331
return = -0.023237118439181942
19 1yr_test 0102-1231
return = -0.31064958811620946
20 1yr_test 0102-1231
return = 1.932614032747914

14-18 training
ema min_vol freq=252 span=63 rf= 0.006161643835616424
weights['AAPL']=-0.0038194253411268
weights['MSFT']=0.0724898120064116
weights['GOOG']=0.6442687392296591
weights['AMZN']=-0.5371248901244982
weights['TSLA']=0.0162001804156728
weights['FB']=0.0750564836390528

weights['BAC']=-0.610737783119529
weights['JPM']=1.3542953132094897
weights['NVDA']=0.0198685264235262
weights['NFLX']=-0.0304969563386583
21 1yr_test 0104-1231
return = 0.5569693162383498
21 1q_test 0104-0331
return = 0.266188811966202
21 2q_test 0401-0630
return = 0.0860071419907636
21 3q_test 0701-0930
return = 0.09742557944747876
21 4q_test 1001-1231
return = -0.030250996235668428
22 1q_test 0103-0331
return = -0.1629198312886773
19 1yr_test 0102-1231
return = 0.5025342284652262
20 1yr_test 0102-1231
return = -0.08676976226578303

14-18 training
ema min_vol freq=252 span=500 rf= 0.006161643835616424
weights['AAPL']=0.1604872104194937
weights['MSFT']=0.139361553609463
weights['GOOG']=0.2364363654561224
weights['AMZN']=-0.1193348192938456
weights['TSLA']=0.0172366735155134
weights['FB']=0.0910914716585399
weights['BAC']=-0.3337054229469052
weights['JPM']=0.9186636550619875
weights['NVDA']=-0.0522217517179207
weights['NFLX']=-0.0580149357624484
21 1yr_test 0104-1231
return = 0.3453276822480061
21 1q_test 0104-0331
return = 0.16158695502636164
21 2q_test 0401-0630
return = 0.05897149537516967
21 3q_test 0701-0930

return = 0.052891050539061193
21 4q_test 1001-1231
return = 0.0030907479670280592
22 1q_test 0103-0331
return = -0.13236128081781068
19 1yr_test 0102-1231
return = 0.5309955754018834
20 1yr_test 0102-1231
return = 0.1962285990855398

16-20 training
ema max_sharpe freq=252 span=63 rf= 0.0032547945205479473
weights['AAPL']=1.0203893997988263
weights['MSFT']=0.283587556397821
weights['GOOG']=0.5978152115432095
weights['AMZN']=0.1665375300591748
weights['TSLA']=1.8829432361718776
weights['FB']=-1.1053955025998883
weights['BAC']=-1.7928998494242272
weights['JPM']=1.831260501772625
weights['NVDA']=-1.3988704261391585
weights['NFLX']=-0.48536765758026
21 1yr_test 0104-1231
return = -0.6750971509730189
21 1q_test 0104-0331
return = -0.33684521753347574
21 2q_test 0401-0630
return = -0.5323571631993279
21 3q_test 0701-0930
return = 0.32196410367746464
21 4q_test 1001-1231
return = 0.3127433695285362
22 1q_test 0103-0331
return = 0.31493158077748584

16-20 training
ema max_sharpe freq=63 span=63 rf= 0.0032547945205479473
weights['AAPL']=0.7188311703292573
weights['MSFT']=-0.0429451191679143
weights['GOOG']=0.3544593539242127

weights['AMZN']=0.0737632244725286
weights['TSLA']=0.57087029876929
weights['FB']=-0.7382258358610672
weights['BAC']=-0.8642493986572577
weights['JPM']=1.2352958885067624
weights['NVDA']=-0.5376846376933373
weights['NFLX']=0.2298850553775252
21 1yr_test 0104-1231
return = -0.15447966659154788
21 1q_test 0104-0331
return = -0.08957936320477468
21 2q_test 0401-0630
return = -0.22612420036534778
21 3q_test 0701-0930
return = 0.14103640315162344
21 4q_test 1001-1231
return = 0.09507545927987247
22 1q_test 0103-0331
return = 0.019910680694537934

16-20 training
ema max_sharpe freq=63 span=30 rf= 0.0032547945205479473
weights['AAPL']=0.5214281450893352
weights['MSFT']=0.1935597684703496
weights['GOOG']=-0.0158426663775441
weights['AMZN']=0.3466565781525864
weights['TSLA']=0.43048323670526
weights['FB']=-0.7956202643360716
weights['BAC']=-0.6554547063489602
weights['JPM']=1.1069482497075047
weights['NVDA']=-0.4085108146439043
weights['NFLX']=0.2763524735814442
21 1yr_test 0104-1231
return = -0.1748227769094163
21 1q_test 0104-0331
return = -0.09717269380451017
21 2q_test 0401-0630
return = -0.2086074151541571
21 3q_test 0701-0930
return = 0.1693927746722016

21 4q_test 1001-1231
return = 0.08467995259623695
22 1q_test 0103-0331
return = 0.013958495336072053

16-20 training
ema max_sharpe freq=252 span=252 rf= 0.0032547945205479473
weights['AAPL']=0.9611588868315643
weights['MSFT']=-1.5533350486760773
weights['GOOG']=0.7051067893095146
weights['AMZN']=0.0093727027276532
weights['TSLA']=2.0
weights['FB']=-0.3730551356741399
weights['BAC']=-1.046350047731837
weights['JPM']=1.13519912758026
weights['NVDA']=-0.4930235672211293
weights['NFLX']=-0.345073707145808
21 1yr_test 0104-1231
return = -0.09330090899881743
21 1q_test 0104-0331
return = -0.3208464040857004
21 2q_test 0401-0630
return = -0.20511587650038604
21 3q_test 0701-0930
return = 0.27481585829441924
21 4q_test 1001-1231
return = 0.475184622489959
22 1q_test 0103-0331
return = 0.08980124210555386

16-20 training
ema max_sharpe freq=252 span=500 rf= 0.0032547945205479473
weights['AAPL']=1.2435503185850143
weights['MSFT']=-1.5420523856955783
weights['GOOG']=0.0485388285835328
weights['AMZN']=0.1079432864076499
weights['TSLA']=2.0000000000000004
weights['FB']=-0.4384688257280196
weights['BAC']=-1.185670399435259
weights['JPM']=1.108912408894983

weights['NVDA']=0.0146682873104793
weights['NFLX']=-0.3574215189228024
21 1yr_test 0104-1231
return = 0.11732843839418422
21 1q_test 0104-0331
return = -0.511235642948019
21 2q_test 0401-0630
return = -0.06564642484989017
21 3q_test 0701-0930
return = 0.2530202416730011
21 4q_test 1001-1231
return = 0.7067888977271802
22 1q_test 0103-0331
return = 0.0957596063171071

16-20 training
ema min_vol freq=252 span=63 rf= 0.0032547945205479473
weights['AAPL']=0.0009549395955211
weights['MSFT']=0.0572016852646759
weights['GOOG']=0.3221154634389186
weights['AMZN']=0.1191108031801427
weights['TSLA']=-0.0128784746624228
weights['FB']=-0.1521184927456592
weights['BAC']=0.0321108324183201
weights['JPM']=0.276752390092991
weights['NVDA']=0.1025316322660134
weights['NFLX']=0.2542192211514993
21 1yr_test 0104-1231
return = 0.4739554162517239
21 1q_test 0104-0331
return = 0.12256424101661798
21 2q_test 0401-0630
return = 0.09492759595040837
21 3q_test 0701-0930
return = 0.07269928724562028
21 4q_test 1001-1231
return = 0.055907026683940084
22 1q_test 0103-0331
return = -0.1179442842407676

16-20 training
ema min_vol freq=63 span=63 rf= 0.0032547945205479473
weights['AAPL']=0.0105590061660762
weights['MSFT']=0.0639760257549297
weights['GOOG']=0.3362271040195446
weights['AMZN']=0.1075233873906431
weights['TSLA']=0.0014114250627991
weights['FB']=-0.175141645522488
weights['BAC']=0.0646031917023839
weights['JPM']=0.2237323922542925
weights['NVDA']=0.127779756822494
weights['NFLX']=0.2393293563493249
21 1yr_test 0104-1231
return = 0.5213786250734976
21 1q_test 0104-0331
return = 0.12084052778557501
21 2q_test 0401-0630
return = 0.10712929456514586
21 3q_test 0701-0930
return = 0.07440468866769644
21 4q_test 1001-1231
return = 0.0802273682923119
22 1q_test 0103-0331
return = -0.10453699277367129

16-20 training
ema min_vol freq=63 span=30 rf= 0.0032547945205479473
weights['AAPL']=0.0288708328640248
weights['MSFT']=0.1008530349305992
weights['GOOG']=0.3440154130951138
weights['AMZN']=0.1069676054101854
weights['TSLA']=0.0387411333139147
weights['FB']=-0.2012315499007373
weights['BAC']=0.097555877182126
weights['JPM']=0.1396220448204289
weights['NVDA']=0.1844258684233291
weights['NFLX']=0.1601797398610155
21 1yr_test 0104-1231
return = 0.6152458422108407
21 1q_test 0104-0331

return = 0.13885104088282305
21 2q_test 0401-0630
return = 0.07027678489939077
21 3q_test 0701-0930
return = 0.07440468866769644
21 4q_test 1001-1231
return = 0.13561814183696919
22 1q_test 0103-0331
return = -0.06964062198727158

16-20 training
ema min_vol freq=252 span=252 rf= 0.0032547945205479473
weights['AAPL']=0.0243666326984878
weights['MSFT']=-0.0396902250153058
weights['GOOG']=0.3869358513105304
weights['AMZN']=0.3890333205767529
weights['TSLA']=-0.03604755846105
weights['FB']=-0.1121291635698079
weights['BAC']=-0.204835164046552
weights['JPM']=0.4886433650771793
weights['NVDA']=-0.110003837851534
weights['NFLX']=0.2137267792812994
21 1yr_test 0104-1231
return = 0.15447186919617234
21 1q_test 0104-0331
return = 0.09627818583227833
21 2q_test 0401-0630
return = 0.024728748280213098
21 3q_test 0701-0930
return = 0.04905300152254454
21 4q_test 1001-1231
return = -0.062452630608110925
22 1q_test 0103-0331
return = -0.10961950772648328

16-20 training
ema min_vol freq=252 span=500 rf= 0.0032547945205479473
weights['AAPL']=0.0471837154487892
weights['MSFT']=-0.0246266022463904
weights['GOOG']=0.3653663875908934

weights['AMZN']=0.4063420628333646
weights['TSLA']=-0.0264238095260919
weights['FB']=-0.0503740717440858
weights['BAC']=-0.2534360867095144
weights['JPM']=0.5275825967027327
weights['NVDA']=-0.1498171754484958
weights['NFLX']=0.1582029830987984
21 1yr_test 0104-1231
return = 0.10609055546093825
21 1q_test 0104-0331
return = 0.09019675159208593
21 2q_test 0401-0630
return = 0.019024107652068255
21 3q_test 0701-0930
return = 0.0388025946388297
21 4q_test 1001-1231
return = -0.07239883317277604
22 1q_test 0103-0331
return = -0.11025855452701076

14-20 training
ema max_sharpe freq=252 span=500 rf= 0.0032547945205479473
weights['AAPL']=1.2260783090495704
weights['MSFT']=-1.5319040867807479
weights['GOOG']=0.0300798607850855
weights['AMZN']=0.1126960118783708
weights['TSLA']=2.0
weights['FB']=-0.4354884204626708
weights['BAC']=-1.195059051511161
weights['JPM']=1.1133925645745903
weights['NVDA']=0.030195963474345
weights['NFLX']=-0.3499911510073824
21 1yr_test 0104-1231
return = 0.12182884984333814
21 1q_test 0104-0331
return = -0.5144259273440545
21 2q_test 0401-0630
return = -0.062278363524344287
21 3q_test 0701-0930
return = 0.2529423691105763

21 4q_test 1001-1231
return = 0.7088627103464983
22 1q_test 0103-0331
return = 0.09118399603748671

14-20 training
ema max_sharpe freq=252 span=1260 rf= 0.0032547945205479473
weights['AAPL']=0.6675518787659517
weights['MSFT']=-0.2927552939292782
weights['GOOG']=-0.6084819445094602
weights['AMZN']=0.3747968433981903
weights['TSLA']=0.9071806554911404
weights['FB']=-0.3577449857772803
weights['BAC']=-0.9209521884848856
weights['JPM']=0.8307094254667521
weights['NVDA']=0.4194872937153678
weights['NFLX']=-0.0197916841364979
21 1yr_test 0104-1231
return = 0.30469278021007634
21 1q_test 0104-0331
return = -0.38576660274329444
21 2q_test 0401-0630
return = -0.09461725956802745
21 3q_test 0701-0930
return = 0.12886534689081275
21 4q_test 1001-1231
return = 0.5239345745938062
22 1q_test 0103-0331
return = -0.03023721043270803

14-20 training
ema max_sharpe freq=252 span=63 rf= 0.0032547945205479473
weights['AAPL']=1.0139471004109508
weights['MSFT']=0.2916341163002874
weights['GOOG']=0.5993882534631642
weights['AMZN']=0.1630864347897697
weights['TSLA']=1.9047857129261607
weights['FB']=-1.0908263601972805
weights['BAC']=-1.828423211133939
weights['JPM']=1.8610124404627268

weights['NVDA']=-1.4296064708986418
weights['NFLX']=-0.4849980161231989
21 1yr_test 0104-1231
return = -0.7064237511867555
21 1q_test 0104-0331
return = -0.3404025999221207
21 2q_test 0401-0630
return = -0.5441644984622046
21 3q_test 0701-0930
return = 0.324586285072383
21 4q_test 1001-1231
return = 0.30458725488636224
22 1q_test 0103-0331
return = 0.30458725488636224

14-20 training
ema min_vol freq=252 span=500 rf= 0.0032547945205479473
weights['AAPL']=0.0487941714304751
weights['MSFT']=-0.0219558619567416
weights['GOOG']=0.3623939552550718
weights['AMZN']=0.4031204007839126
weights['TSLA']=-0.0263611415804291
weights['FB']=-0.0482930415920741
weights['BAC']=-0.254228360785406
weights['JPM']=0.528662978192055
weights['NVDA']=-0.1487784398586336
weights['NFLX']=0.1566453401117701
21 1yr_test 0104-1231
return = 0.10755429719255329
21 1q_test 0104-0331
return = 0.09006263560542604
21 2q_test 0401-0630
return = 0.019557143777747407
21 3q_test 0701-0930
return = 0.03869134709547041
21 4q_test 1001-1231
return = -0.07142545357173448
22 1q_test 0103-0331
return = -0.11059646616032975

14-20 training
ema min_vol freq=252 span=1260 rf= 0.0032547945205479473
weights['AAPL']=0.1017541876901216
weights['MSFT']=0.0455054317421859
weights['GOOG']=0.2959600771306538
weights['AMZN']=0.2947749927307899
weights['TSLA']=-0.0137479491234959
weights['FB']=0.0111341960617466
weights['BAC']=-0.2799749739612836
weights['JPM']=0.5733574504913332
weights['NVDA']=-0.1185663729613703
weights['NFLX']=0.0898029601993186
21 1yr_test 0104-1231
return = 0.16332373110413131
21 1q_test 0104-0331
return = 0.09006263560542604
21 2q_test 0401-0630
return = 0.037360996096216494
21 3q_test 0701-0930
return = 0.036032733320184404
21 4q_test 1001-1231
return = -0.03872489508981118
22 1q_test 0103-0331
return = -0.11441714991191798

14-20 training
ema min_vol freq=252 span=63 rf= 0.0032547945205479473
weights['AAPL']=-0.0004917552426269
weights['MSFT']=0.0573776300594717
weights['GOOG']=0.3218634989518522
weights['AMZN']=0.1189698966144739
weights['TSLA']=-0.0136087142671911
weights['FB']=-0.1504558519573091
weights['BAC']=0.0323333537745695
weights['JPM']=0.2764571555526009
weights['NVDA']=0.1027687095092875
weights['NFLX']=0.2547860770048714
21 1yr_test 0104-1231
return = 0.47382540585200605
21 1q_test 0104-0331

return = 0.12283718184621818
21 2q_test 0401-0630
return = 0.09507928551253607
21 3q_test 0701-0930
return = 0.07255575887759355
21 4q_test 1001-1231
return = 0.05537593573433347
22 1q_test 0103-0331
return = -0.11859322820506082