COLUMBIA UNIVERSITY

Time Series Forecasting and Portfolio Optimization of Magnificent 7 Stocks

Final Project Report for STAT 5261 - Spring 2024

Team members:

1. Sanchit Kabra (ssk2278)
2. Rhonish Nair (rn2563)
3. Ziv Chu (tc3310)
4. Yulan Ma (ym3016)
5. Sihan Wang (sw3853)
6. Spencer Qu (yq2373)

1. Introduction

In today's ever-changing financial markets, understanding and forecasting trends in the stock market is crucial for making investment decisions. Also, portfolio optimization is significant in maximizing investment returns and minimizing risks associated with market fluctuations. In this project, we investigate the dynamics and correlations within a selected portfolio of technology stocks over three years, using advanced econometric methods to forecast stock trends and optimize investment strategies. This systematic approach enables us to effectively manage risks while also seizing market opportunities, thereby improving the overall performance of the portfolio.

The study leverages historical stock price data from seven leading technology firms—Apple, Google, Microsoft, Amazon, Nvidia, Meta and Tesla over the past three years from April 1st, 2021 to April 1st, 2024, obtained through Yahoo Finance. We mainly focus on daily closing prices to compute the daily returns to improve the accuracy of the data.

2. Objective

The primary objective of the project is to develop a new approach to constructing and optimizing investment portfolios and to test its plausibility by implementing time series modeling and risk management. By applying these techniques, we aim to enhance the predictive accuracy and risk management strategies for an investment portfolio comprising the leading seven technology stocks. In our study, we build the best-performing portfolios in terms of the Sharpe ratio.

Initially, we calculate log returns to normalize the price movements and provide a clearer view of growth over time relative to the starting prices. Then we would like to find the correlations between seven stocks. The optimal portfolio weighting method is based on past performance compared to NASDAQ 100 holding returns, choosing from equal-weighted, value-weighted, and mean-variance methods. After obtaining the final optimal portfolio, we use the time series model to do the future prediction. We would like to test whether the VAR, ARIMA, SARIMA or GARCH model works best for our data, and predict future stock return with the model we have chos

3. Methodology

3.1 Stock Selection

We used Yahoo Finance to find 7 notable technology companies, who are Apple, Google, Microsoft, Amazon, NVIDIA, Meta, and Tesla. We achieved their daily stock price for the past three years from now and we use two different general plots to compare their close price (Graph 3.1.2). In order to get more information, we also visualized each individual detailed plot for each stock so that we can see their general trending changes from the 30-day SMA and closing prices over the 3-year period. (Graph 3.1.3).

In order to build our time series model for prediction, we transformed the stock close price into log return to ensure that all data are stationary for time series model building (Code 3.1). Also, we found that all of these companies' stocks are highly correlated (Graph 3.1.1)., which might be caused by general market conditions and trending. With these basic concepts visualized, we can proceed to the next step of building our time series model with reference to the information above.

3.2 Time Series Model Selection

We selected the Arima model for forecasting at first (Code 3.2.1). But since Arima's forecast result isn't favorable, we added the seasonal facts into Arima, which is the Sarima model (Code 3.2.2). We used the Auto Arima package in python to choose the model with the best AIC and we used ACF plots to find proper p values (Code 3.2.3). From the ACF and PACF plots, we can find that the most proper p-values should be 0 or 1 since there are no significant spikes (Graph 3.2).

Also, due to the high correlation between these companies, the VAR model, whose nature is designed for highly correlated data, is used here for forecasting (Code 3.2.4). Moreover, we built a GARCH model for testing variations. The basic GARCH model had a relatively low MSE, but the prediction results are similar to our Arima model, which we decided not to proceed with this model.

3.3 Testing and Forecasting

We split the data for 20 days as a test group and forecast 40 days beyond the train data so that 20 days for testing and 20 days' forecasted data.

For the testing part, MSE is the main index we used to compare the three models. Also, we visualized the three models' forecasted log returns in the test group to compare the accuracy and decide the forecasted data for later computation.

3.4 Portfolio Weight Allocation

The method we chose to optimally allocate weights to our portfolio was Mean-Variance optimization. Our optimization function was the Sharpe's ratio with the basic constraints that the individual weights add up to 1 for the portfolio and that no short selling or leveraging is allowed i.e. no negative weight allocation. The Python library used to perform these calculations was Scipy and the specific method used is called the Sequential Least Squares Programming (SLSQP) method, which is appropriate for this kind of constrained nonlinear optimization problem. (Code 3.4.1)

This whole step was carried out for each rolling 5-day window forecast for a total of 74 rolling windows. This takes into account the constantly changing stock movement and therefore updates every business day to create the most optimal portfolio. (Code 3.4.2)

3.5 Portfolio Performance

As a result of setting our objective to optimize against maximizing the Sharpe ratio for our portfolio's weights, we were able to see the outperformance of our optimized portfolio (*blue*) in comparison to both the Nasdaq 100 (*green*) and a hypothetically equally weighted portfolio (*orange*) (Figure 4.5.1). In an equally weighted portfolio, we weighted each stock with a weight of 1/7 (~14.28%). Meanwhile, since our investment universe contained the seven most weighted stocks in the Nasdaq 100, it makes sense to compare the performance of our portfolio against the benchmark containing these seven stocks.

4. Results

4.1 VAR vs ARIMA vs SARIMA

The predicted value versus the actual value in the test time zone is plotted. The log return predicted and ARIMA and SARIMA are both 0 for this time period (GRAPH 4.1.1). This is due to low p-value and short test time range, which limits the prediction for trend. We would only put the Arima model's predictions in this report since the predictions of them are exactly the same.

Compared to the ARIMA models, the VAR model shows little trend in each day (GRAPH 4.1.2). However, it is close to 0 as well. Also, The mse bar plot shows very little difference between the Arima models and Var models (Graph 4.1.3). By combining the correlated nature of 7 companies and the low performance of Arima that all predictions are 0, we chose VAR as our final model and used its predicted log return for later computation.

4.2 Optimal Portfolio Allocation

The average portfolio allocation over the 74 rolling window period was heavily skewed towards Nvidia which took 71% weight, followed by Microsoft at 9%. This can be clearly seen in Figure 4.4.1 below. This behavior can be explained by Nvidia's transition from gaming to AI in the last couple of years which has led to a significant boost in their stock returns (Figure 4.4.1).

4.3 Performance Summary

As shown in Figure 4.5.1, our optimized portfolio was far less volatile than the Nasdaq and the equally-weighted portfolio. This is due to the optimization of the Sharpe ratio as we are able to minimize risk and maximize performance during periods of underperformance by certain stocks which is not accomplished via the equally-weighted portfolio (Code 4.3.1).

Furthermore, our portfolio experienced no drawdown periods during our backtest (Figure 4.3.2). This is heavily indicative of the average weight of ~71% to NVIDIA as NVIDIA individually performed best over this time period. (Code 4.3.2) Meanwhile, both the Nasdaq and equally weighted portfolio experienced a maximum drawdown of 5.83% and 5.79% respectively. The two portfolios also had more positive performance than negative performance days, while our optimized portfolio never experienced a period of underperformance (Code 4.3.3). Lastly, both the

optimized and equally weighted portfolios were quite sensitive to the Nasdaq 100 with betas of 0.8 and 0.64 respectively (Code 4.3.4). This is expected given our universe contains the seven most weighted names in the Nasdaq 100 and we measured our beta against the Nasdaq 100

Performance Stats					
	Average Annualized Return	**Max Drawdown**	**No. of Positive Days**	**No. of Negative Days**	**Beta to NDX**
Optimized Wgts	43.68%	0	74	0	0.8
Equal Wgts	25.29%	-5.83%	44	30	0.64
Nasdaq 100	24.25%	-5.79%	54	19	

5. Conclusion

We acknowledge the suboptimal performance of the time series models for these stocks. The model requires further refinement, and deep learning could be a promising approach for enhancing accuracy in future analyses.

Our methodical approach to portfolio optimization, through the use of Mean-Variance optimization, proved effective. Our optimized portfolio consistently outperformed the benchmarks, by getting 1.7 times the returns of NASDAQ in the same period, and showcasing less volatility and no drawdown periods. Further improvements can be made by including other asset classes such as bonds and commodities to mitigate risk while maintaining stable returns during the recessionary periods. Machine learning can also be potentially used to offer better risk-adjusted returns compared to traditional Mean-Variance optimization.

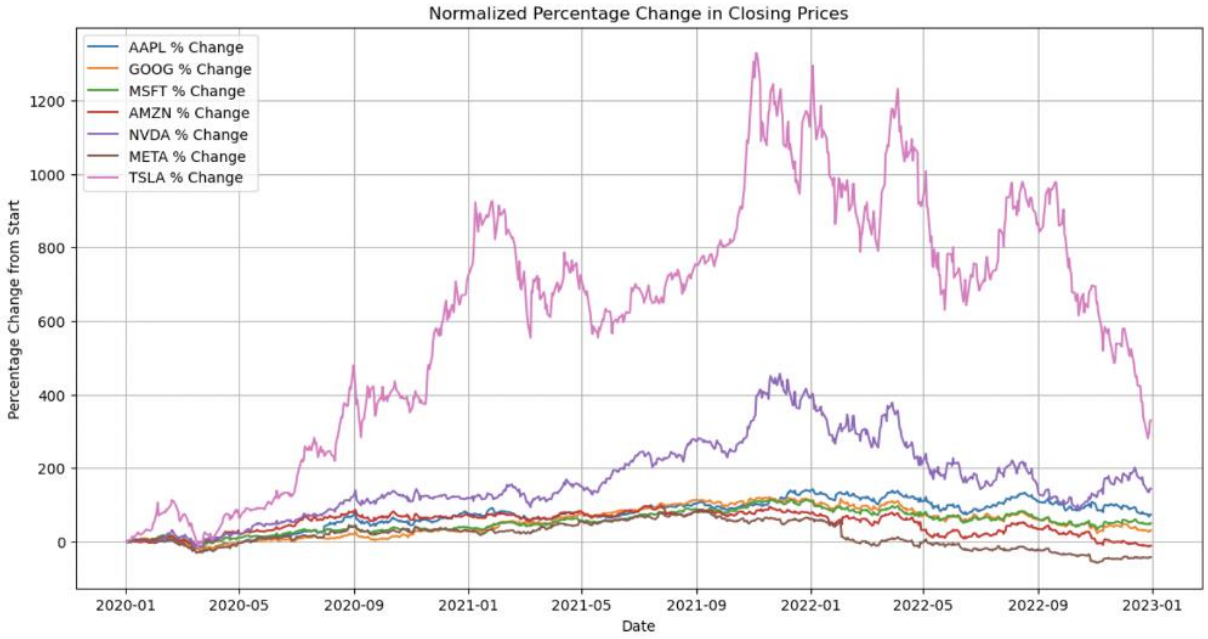
6. Appendix

Graphs and Visualizations

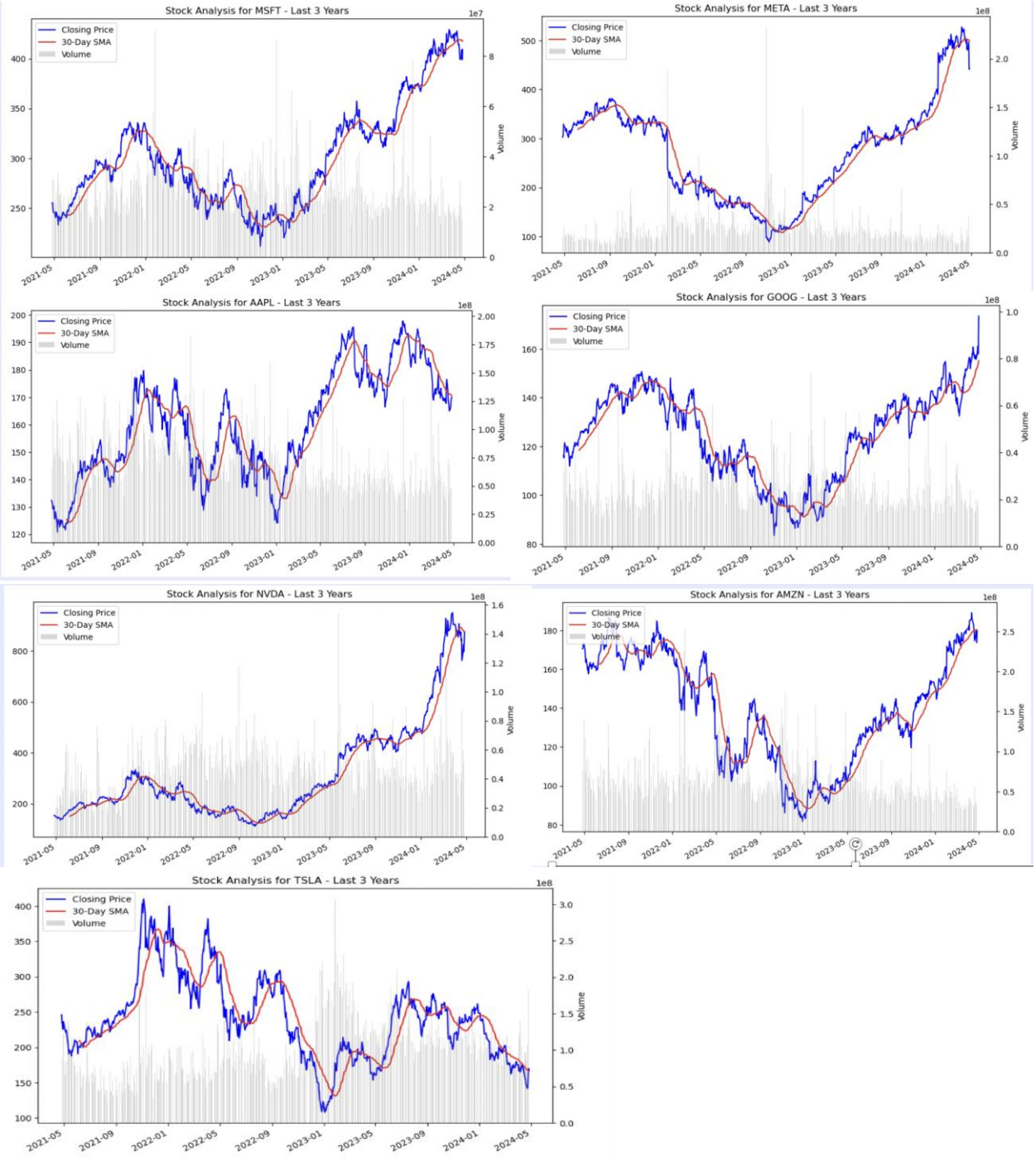
Graph 3.1.1 Correlation table:

	AAPL	GOOG	MSFT	AMZN	NVDA	META	TSLA
AAPL	1.000000	0.664912	0.708172	0.592020	0.595812	0.503484	0.514859
GOOG	0.664912	1.000000	0.716425	0.662887	0.584090	0.600397	0.398663
MSFT	0.708172	0.716425	1.000000	0.670694	0.665232	0.570043	0.422512
AMZN	0.592020	0.662887	0.670694	1.000000	0.592127	0.586903	0.429583
NVDA	0.595812	0.584090	0.665232	0.592127	1.000000	0.501208	0.495046
META	0.503484	0.600397	0.570043	0.586903	0.501208	1.000000	0.319451
TSLA	0.514859	0.398663	0.422512	0.429583	0.495046	0.319451	1.000000

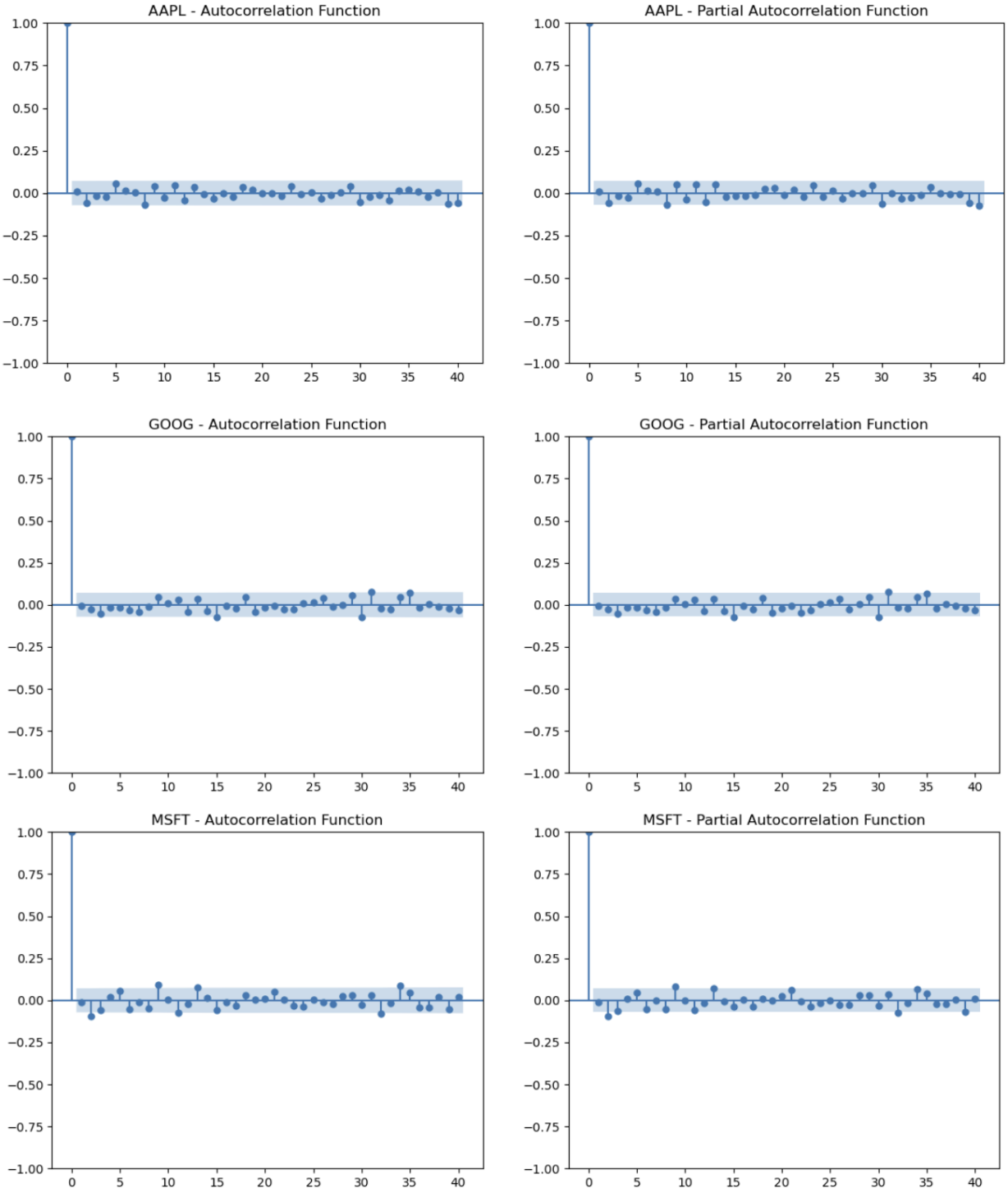
Graph 3.1.2 General Stock Price Visualization:

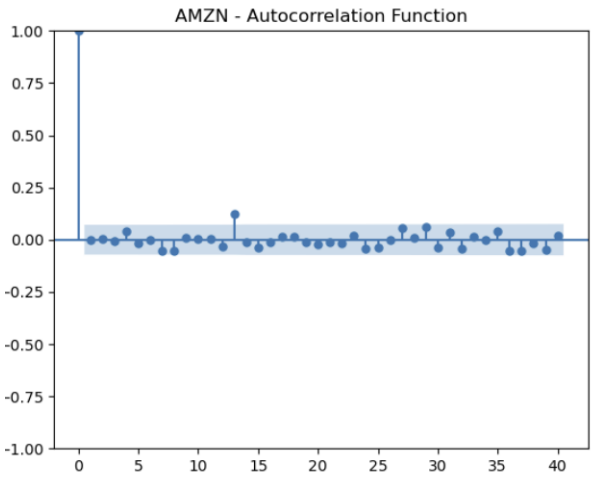


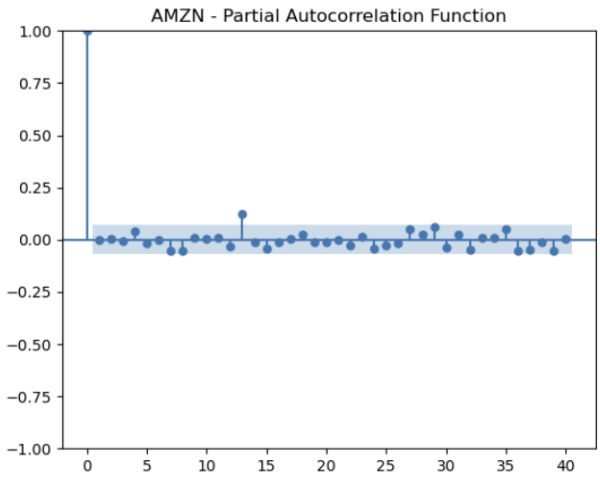
Graph 3.1.3 Individual Stock Price Visualization:

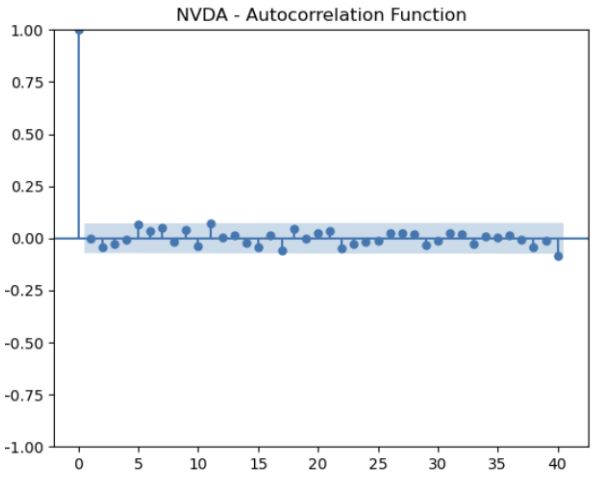

Stock Analysis for MSFT - Last 3 Years
Stock Analysis for META - Last 3 Years
Stock Analysis for AAPL - Last 3 Years
Stock Analysis for GOOG - Last 3 Years
Stock Analysis for NVDA - Last 3 Years
Stock Analysis for AMZN - Last 3 Years
Stock Analysis for TSLA - Last 3 Years
Closing Price
30-Day SMA
Volume

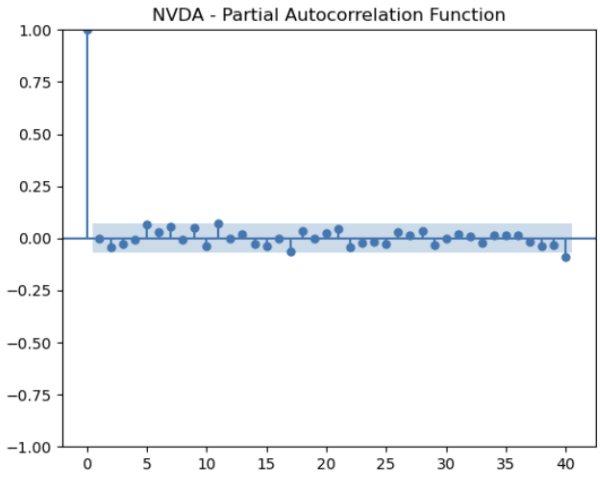
Graph 3.2 ACF and PACF curves

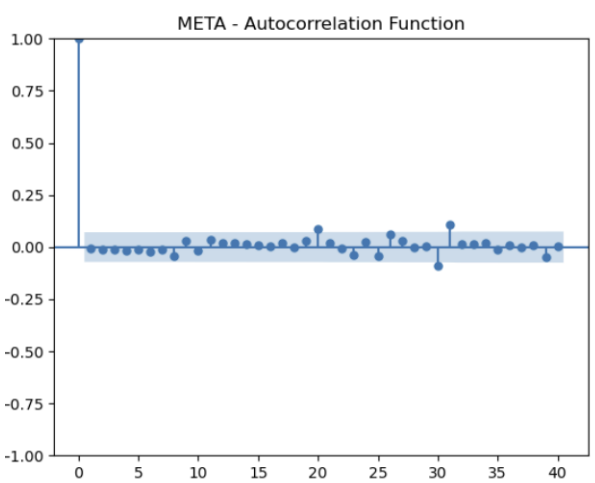


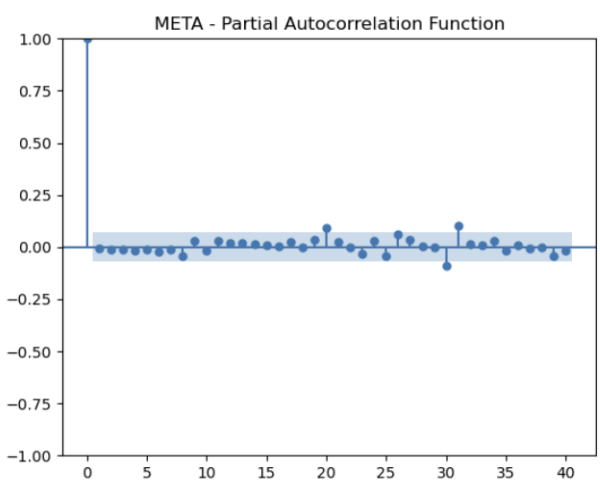

AMZN - Autocorrelation Function
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
0
5
10
15
20
25
30
35
40

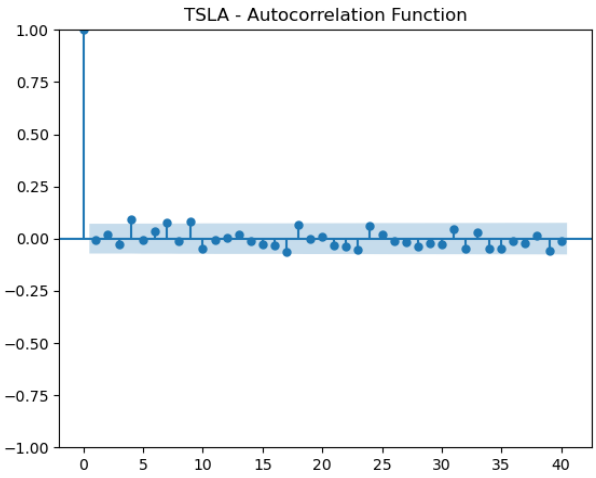

AMZN - Partial Autocorrelation Function
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

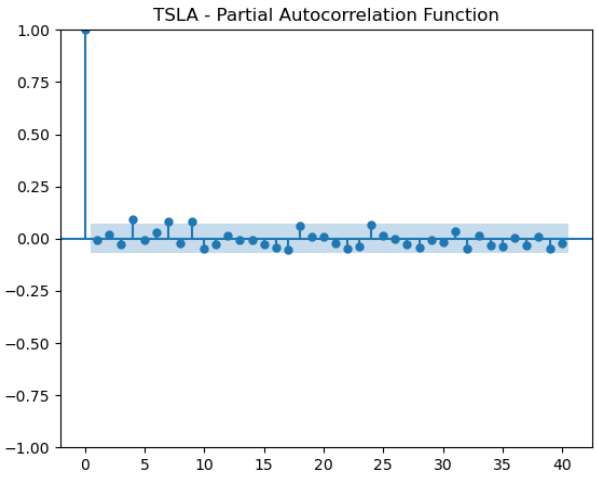

NVDA - Autocorrelation Function
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
0
5
10
15
20
25
30
35
40


NVDA - Partial Autocorrelation Function
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

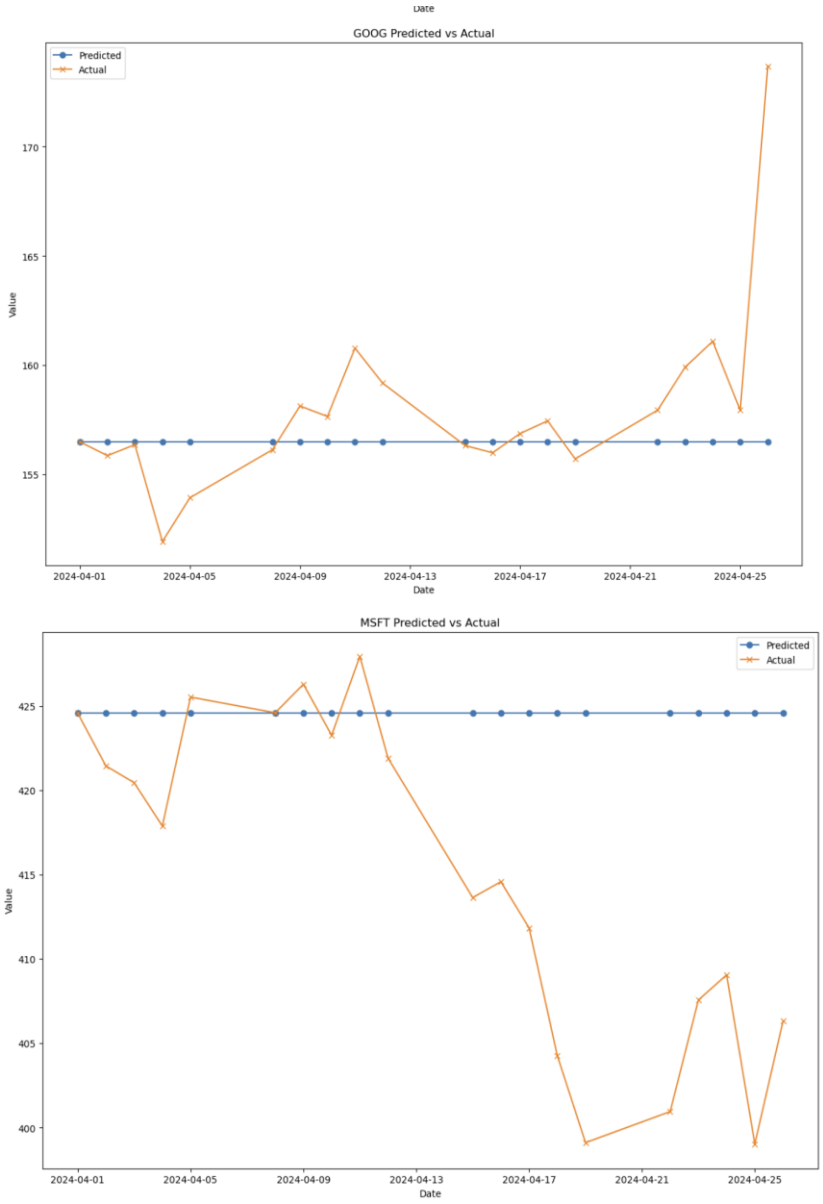

META - Autocorrelation Function
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
0
5
10
15
20
25
30
35
40


META - Partial Autocorrelation Function
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

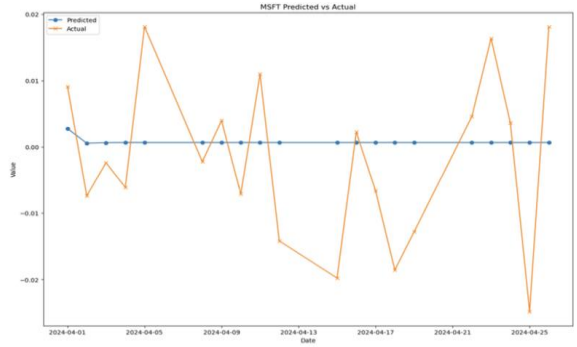

TSLA - Autocorrelation Function
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

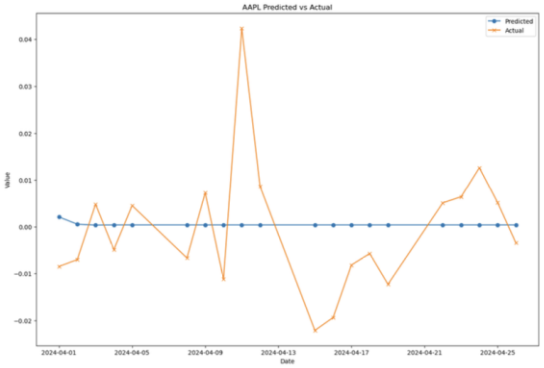

TSLA - Partial Autocorrelation Function
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

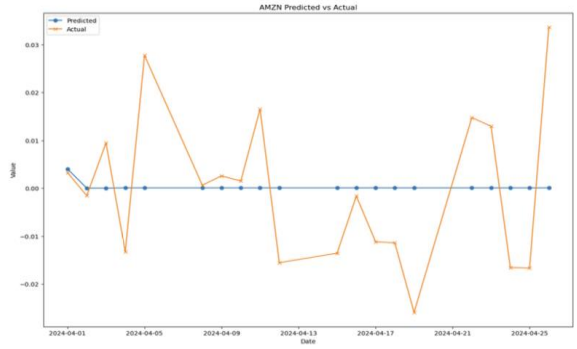
GRAPH 4.1.1

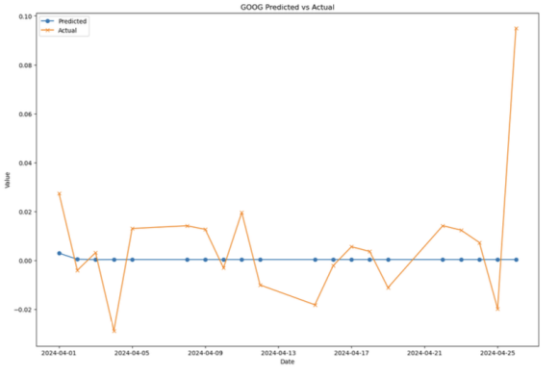

Date
GOOG Predicted vs Actual
Predicted
Actual
170
165
160
155
Value
2024-04-01
2024-04-05
2024-04-09
2024-04-13
2024-04-17
2024-04-21
2024-04-25
Date
MSFT Predicted vs Actual
Predicted
Actual
425
420
415
410
405
400
Value
2024-04-01
2024-04-05
2024-04-09
2024-04-13
2024-04-17
2024-04-21
2024-04-25
Date

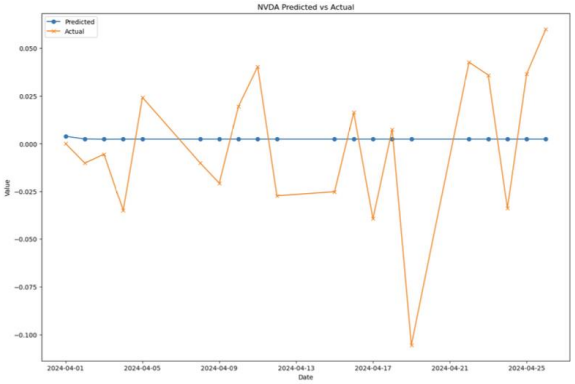
GRAPH 4.1.2

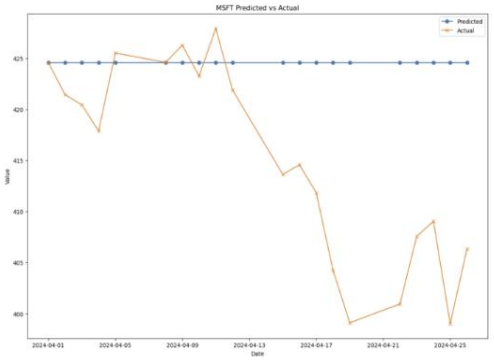

MSFT Predicted vs Actual
Predicted
Actual
Value
Date

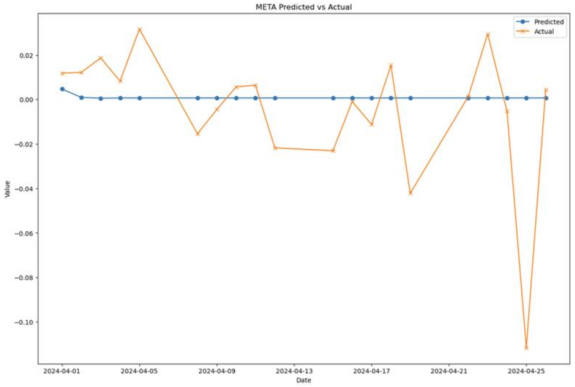

AAPL Predicted vs Actual
Predicted
Actual
Value
Date

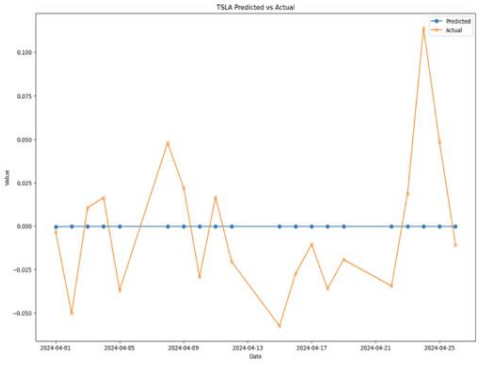

AMZN Predicted vs Actual
Predicted
Actual
Value
Date


GOOG Predicted vs Actual
Predicted
Actual
Value
Date


NVDA Predicted vs Actual
Predicted
Actual
Value
Date


MSFT Predicted vs Actual
Predicted
Actual
Value
Date


META Predicted vs Actual
Predicted
Actual
Value
Date


TSLA Predicted vs Actual
Predicted
Actual
Value
Date

GRAPH 4.1.3

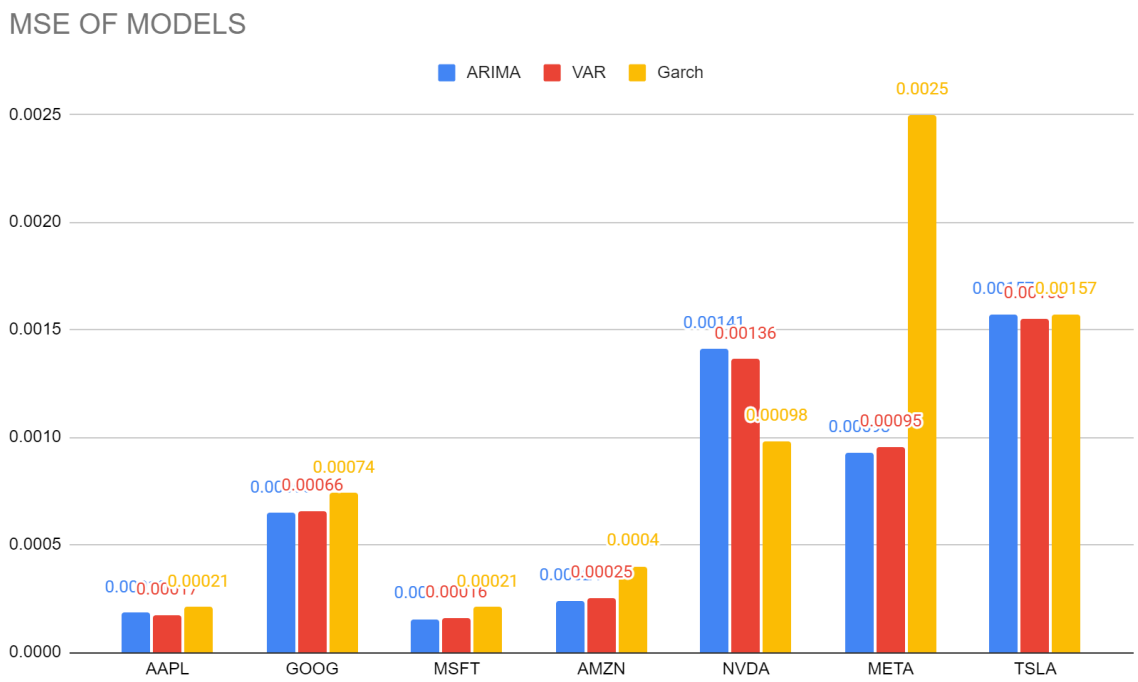


Figure 4.3.1

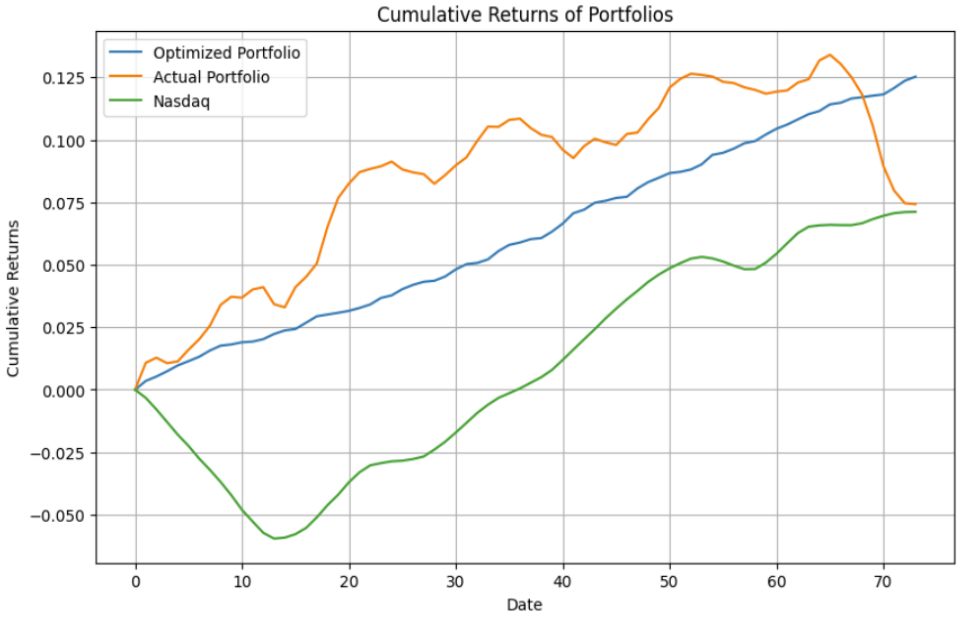


Figure 4.3.2

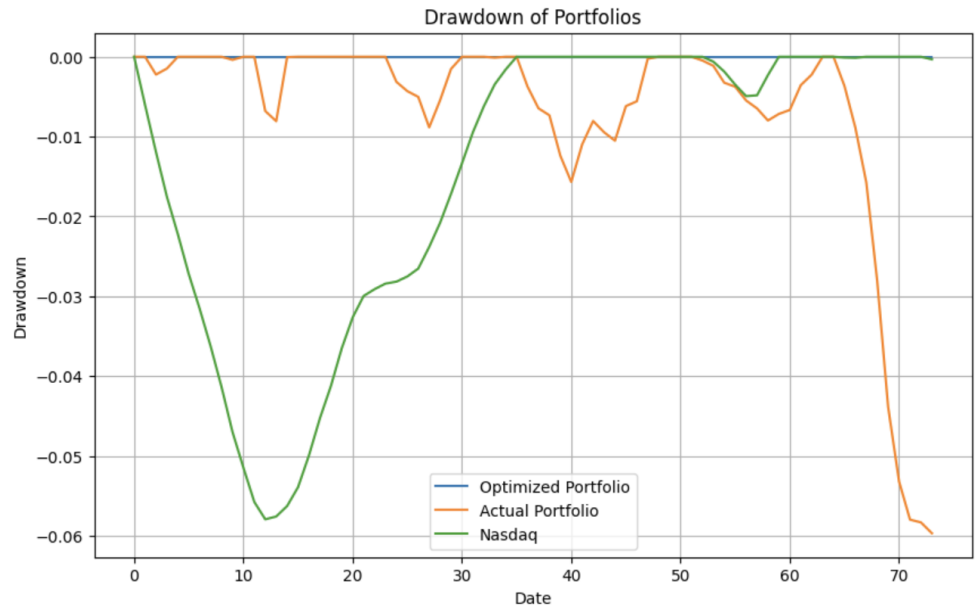
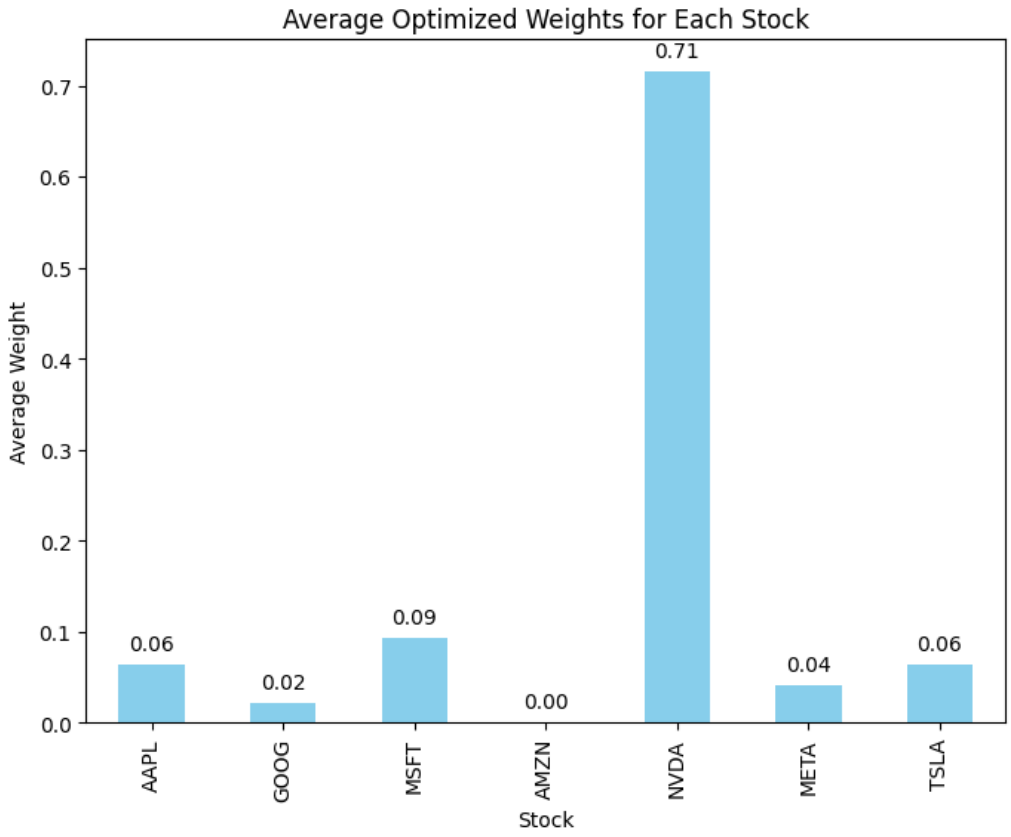


Table 4.3.1

Performance Stats					
	Average Annualized Return	**Max Drawdown**	**No. of Positive Days**	**No. of Negative Days**	**Beta to NDX**
Optimized Wgts	43.68%	0	74	0	0.8
Equal Wgts	25.29%	-5.83%	44	30	0.64
Nasdaq 100	24.25%	-5.79%	54	19	

Figure 4.4.1



ARIMA - SARIMA

Ex: Figure 4.X.1, Figure 3.X.2, etc.

Code

Code 3.1

```
import yfinance as yf
```

```
def calculate_log_return(df, column_name='Close'):
    # Calculate the log returns
    log_returns = np.log(df[column_name] / df[column_name].shift(1))
    return log_returns
def Get_hist(name,year):
    data = yf.Ticker(name)
    hist = data.history(period = year, interval='1d')
    df = pd.DataFrame(hist)
    df = df.dropna()
    df['Log_R'] = calculate_log_return(df)
    df = df.fillna(0)
    return df['Log_R'].reset_index(),df['Close'].reset_index()
```

```
tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN', 'NVDA', 'META', 'TSLA']
data = {}
close = {}
for name in tech_list:
    data[name],close[name]=Get_hist(name,"3y")
```

Code 3.2.1 (Arima Model)

```
model = auto_arima(df['Log_R'],seasonal=False,
                   trace=True,
                  error_action='ignore',
                  suppress_warnings=True,
                  stepwise=True)
return model
```

Code 3.2.2 (SARIMA Model)

```
model = auto_arima(df['Log_R'],seasonal=True, m = 14,
                   trace=True,
                  error_action='ignore',
                  suppress_warnings=True,
                  stepwise=True)
return model
```

Code 3.2.3 ACF, PACF plots

```
for company in tech_list:
    df = data[company]  # Get the DataFrame for the company
    df.dropna(subset=['Log_R'], inplace=True)  # Ensure no NaN values in 'Log_R'

    # Create figure for ACF and PACF plots
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # ACF plot
    plot_acf(df['Log_R'], ax=axes[0], lags=40, title=f'{company} - Autocorrelation Function')

    # PACF plot
    plot_pacf(df['Log_R'], ax=axes[1], lags=40, title=f'{company} - Partial Autocorrelation Function')

    plt.show()
```

Code 3.2.4 (VAR model)

```
train, test = merged_df[:-20], merged_df[-20:]
modelvar = VAR(train)
v_results = modelvar.fit(maxlags =1 )
print(v_results.summary())
```

Code 3.4.1

```
def sharpe_ratio(weights, means, cov, risk_free_rate):
    portfolio_return = np.dot(weights, means)
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov, weights)))
    return -(portfolio_return - risk_free_rate) / portfolio_volatility

def optimize_portfolio(returns):
    means = np.mean(returns, axis=0)
    cov = np.cov(returns, rowvar=False)
    num_assets = returns.shape[1]
    constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})  # Weights must sum to 1
    bounds = tuple((0, 1) for asset in range(num_assets))
    initial_guess = np.array(num_assets * [1. / num_assets])
    result = minimize(sharpe_ratio, initial_guess, args=(means, cov, risk_free_rate),
                      method='SLSQP', bounds=bounds, constraints=constraints)
    return result.x
```

Code 3.4.2

```
# Optimize portfolio for the forecast
optimized_weights = optimize_portfolio(forecast)
all_weights.append(optimized_weights)
forecast_mean = np.mean(forecast, axis=0)
forecast_cov = np.cov(forecast, rowvar=False)
```

Code 4.3.1

```
def calculate_cumulative_returns(portfolio_returns):
    cumulative_returns = portfolio_returns.cumsum()
    cumulative_returns.iloc[0] = 0
    cumulative_returns.name = 'Cumulative Returns'
    return cumulative_returns
```

Code 4.3.2

```
def calculate_drawdown(daily_returns):
    # Convert daily returns to cumulative returns
    cumulative_returns = (1 + daily_returns).cumprod()

    # Calculate the previous peaks
    previous_peaks = cumulative_returns.cummax()

    # Calculate drawdown as the difference between cumulative returns and previous peaks
    drawdown = (cumulative_returns - previous_peaks) / previous_peaks

    return drawdown
```

Code 4.3.3

```
###Number of Positive and Negative Days
def count_positive_negative_days(daily_returns):

    # Count the number of positive and negative days
    positive_days = (daily_returns > 0).sum()
    negative_days = (daily_returns < 0).sum()

    return positive_days, negative_days

positive_days_opt, negative_days_opt= count_positive_negative_days(pd.Series(optimized_means))
positive_days_eq, negative_days_eq = count_positive_negative_days(pd.Series(actual_means))
positive_days_ndx, negative_days_ndx = count_positive_negative_days(pd.Series(nasdaq_rolling_means))
```

Code 4.3.4

```
def calculate_beta(portfolio_returns, benchmark_returns):
    # Calculate covariance and variance
    covariance = np.cov(portfolio_returns, benchmark_returns)[0, 1]
    variance = np.var(benchmark_returns)

    # Calculate beta
    beta = covariance / variance

    return beta
```