FINA 6339: GROUP PROJECT.

Title: Portfolio Optimization & Forecasting

Team Members:

Goyal, Abhishek

Priyam Deepak Pandya

Yu Chen

Introduction.

In a financial market full of uncertainty, portfolio optimization, and forecasting is the core issue in the field of investment management. By building a portfolio that is both resilient to market volatility and capable of sustained growth, individual investors and institutions can more effectively plan their assets and future investment strategies. This study aims to explore a variety of methods for creating and forecasting optimal portfolios and to propose a framework that is both scientific and practical to optimize portfolios. First, our study starts with stock selection. Using the data of "ALLY", "AMZN", "AXP", "AON", "AAPL", "BATRK", "BAC", "BYDDF", "COF", "CHTR" for the past five years, this paper will analyze their returns and covariance and other statistics. Explore how to pick stocks during the initial portfolio creation phase. The analysis of this stage not only lays the foundation for the subsequent optimization but also provides a starting point for investors to conduct empirical analysis. Secondly, in the optimization part, mean-variance, Black-Letterman, Monte Carlo, and other models will be used to optimize the weight of the initially selected stock portfolio. These models analyze the risks and returns of a portfolio from different perspectives to identify the optimal mix under given conditions. Comparing the optimization results of these models can not only show diversified investment perspectives for investors but also reveal the multi-dimensional characteristics of portfolio management in different market environments. Finally, the forecasting section is dedicated to predicting the future returns of stocks using ARIMA and Long Short-Term Memory (LSTM) models. Through these two methods, this study aims to provide a quantitative forecast of the future performance of stocks to assist investors in making more informed decisions. This paper not only focuses on the quantitative analysis of portfolio construction but also pays more attention to predicting and analyzing future trends. The research aims to provide investors with an empirical, integrated portfolio management strategy that enables them to find a balance between robustness and growth in a changing market. Through three specific analytical sections - selection, optimization, and forecasting - this article will guide readers through the complexities of quantitative investment management and show how these strategies can be applied in the real world.

Data and Methodology.

1. Choosing Stocks

"Don't put all your eggs in one basket," is the "eggs and basket" theory of "risk diversification" developed by Nobel Prize-winning economist James Tobin. The basic meaning of this sentence is that in investment and financial management, products with risk differences are selected to carry out combinatorial asset allocation, and risks are dispersed to achieve the purpose of reducing the volatility of the overall investment return. But how to do risk diversification, and what not to put eggs in one basket, is something we will discuss.

The rate of return reflects the change in the value of the investment. For a portfolio, the rate of return is a key measure of its performance and can tell investors whether their investment has increased or decreased in value over the past period. A high rate of return may indicate the effectiveness of an investment strategy, while a low rate of return may indicate the need to adjust the strategy. By comparing the rate of return of different assets over the same period, we can determine whether there is a relationship between assets.

Covariance measures the overall trend of change between two random variables, such as the return of two stocks. If the covariance of two variables is positive, then they will usually rise or fall together; If the covariance is negative when one variable goes up, the other will usually go down. In portfolio management, covariance is a key factor in building an effectively diversified portfolio because covariance helps investors understand how different assets in a portfolio relate to each other. By choosing assets with a low or negative covariance, you can reduce the overall risk of your portfolio because it means that asset prices are less likely to move in the same direction at the same time.

The relative Strength Index (RSI) is a momentum indicator that typically has a value range of 0 to 100. It measures recent price changes to determine whether an asset is overbought or oversold. This indicator is used to identify overbought or oversold conditions in a stock to predict future price movements. When the RSI is greater than 70, it may indicate that the stock is overbought and therefore may be at risk of a price correction or decline. When the RSI is below 30, it may indicate that the stock has been oversold, so there may be an opportunity for a price recovery.

We will now discuss whether the following ten stocks which we chose randomly are fitted to be “put in the same basket” by comparing returns, RSI, and covariance.

❿ "ALLY": Ally Financial Inc. - A company offering digital financial services, including auto financing, online banking, mortgage loan services, and more.

❿ "AMZN": Amazon Inc. - A globally renowned online retailer and cloud computing services provider.

❿ "AXP": American Express Company - A financial services company that offers credit card services and global payment solutions.

❿ "AON": Aon plc - A global professional services firm that provides risk, retirement, and health consulting services.

❿ "AAPL": Apple Inc. - A well-known company specializing in consumer electronics, computer software, and online services.

❿ "BATRK": Liberty Braves Group - A company that owns the Atlanta Braves baseball team and related assets.

❿ "BAC": Bank of America Corp - A multinational banking and financial services company offering investment banking, wealth management, and other services.

❿ "BYDDF": BYD Company Limited - A Chinese automobile manufacturer that also ventures into battery production and other electronic products.

- ❿ "COF": Capital One Financial Corp - A company providing financial products such as credit cards, savings accounts, and banking services.
- ❿ "CHTR": Charter Communications, Inc. - A company providing broadband communication services in the United States.

Returns

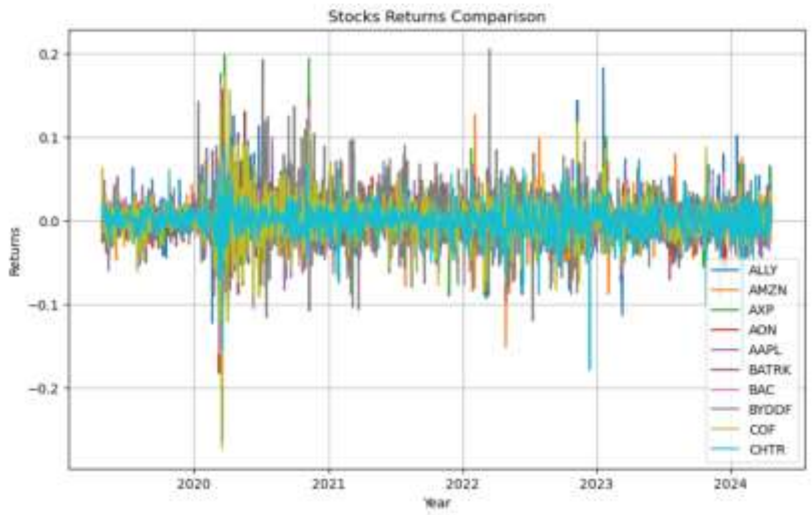


Figure 1

Through the return fluctuation chart, we can find that the income fluctuation patterns of Bank of America (BAC), American Express (AXP), and COF are similar, and these companies are all in the financial field. Apple (AAPL) in the technology sector and BYD Co., LTD. (BYDDF) in the automotive sector show different patterns of volatility, respectively. This means that these companies respond similarly to different circumstances. This suggests that including companies from the same sector might not effectively diversify risk due to similar return movements.

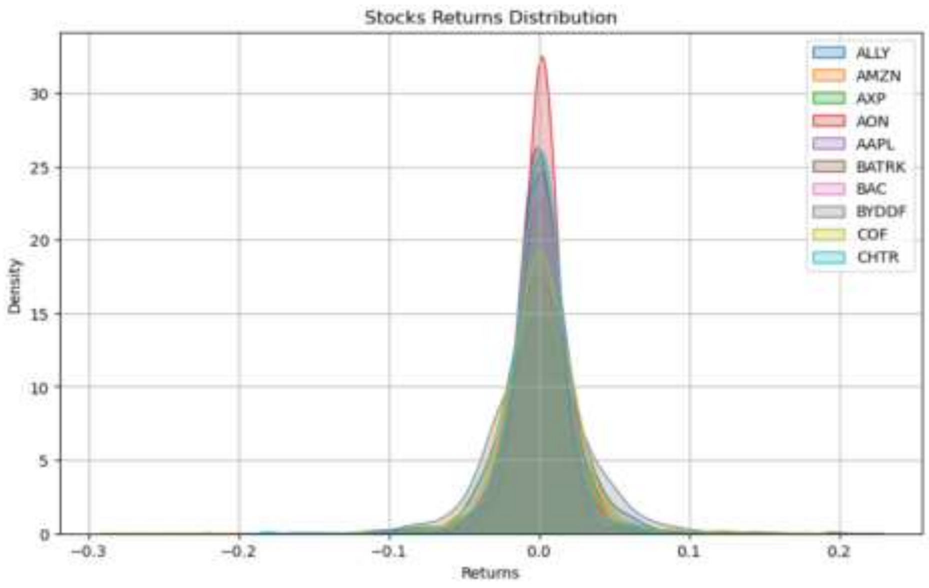


Figure 2

Something similar can be observed from the distribution of returns. The yield distributions of ALLY, BATRK, AXP, and AAPL have nearly converged over the past five years, and ALLY, BATRK, and AXP are all financial companies. AON has the largest peak and CHTR has the smallest peak.

Relative Strength Index

In portfolio management, RSI can be used as a tool to measure the current price dynamics of a portfolio or individual assets within it, helping investors decide when to buy or sell. If two stocks have similar RSI values, this may indicate that they have shown similar buying and selling strength over the same period. This similarity can be caused by a variety of factors, such as they may react similarly to certain changes in the market, or they may be affected by similar market sentiment in the same industry. Therefore, RSI can also help determine the correlation between assets.

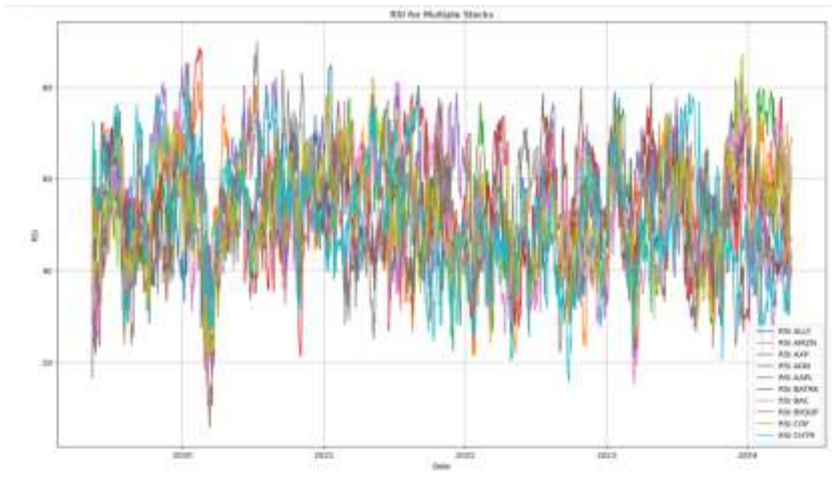


Figure 3

We can observe that the RSI values of AXP, BAC, and COF show different fluctuation patterns in different periods. However, their RSI volatility trends are very close, which may mean that they have similar reactions to market events, which is a point to be aware of in portfolio risk diversification.

Risk

As we know, the higher the variance of a stock is the higher the risk, and the higher the covariance between two stocks is the lower the risk diversification. The greater the covariance, the greater the correlation between two stocks. Correlation is used to describe the relationship between asset returns. If the yields of two assets are highly correlated, they tend to move in similar ways under similar market conditions. Stock correlations range from -1 to +1, where:

+1: indicates a perfect positive correlation, meaning that when the yield of one asset rises, the yield of the other asset will also rise by the same percentage.

Zero: indicates no correlation

-1: indicates a perfect negative correlation, that is, when the yield of one asset rises, the yield of the other asset will fall by the same proportion.

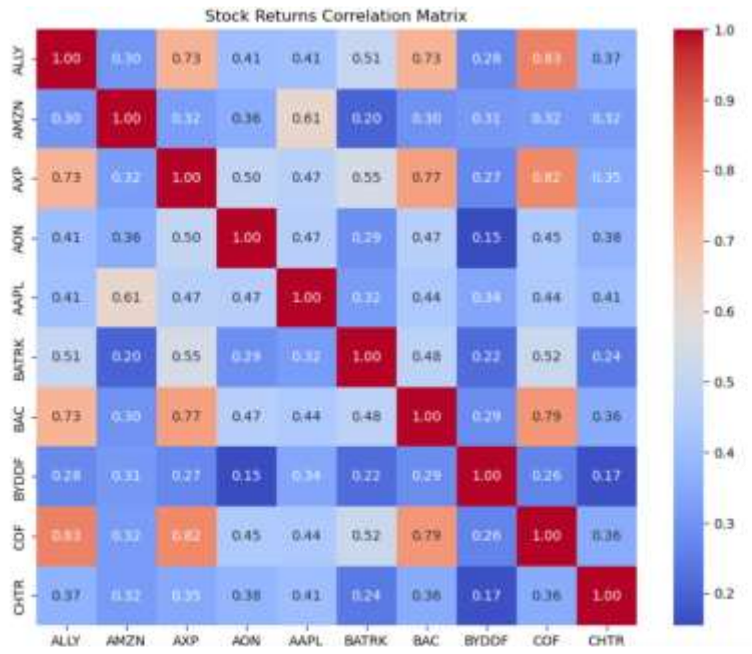


Figure 4

According to the correlation coefficient chart, we know that these ten stocks are positively correlated with each other, and some stocks have a strong positive correlation with each other. So, we need further diversification to optimize risk management. In this case, we may wish to consider diversifying risk by adding some stocks with low correlation coefficients, or by reducing those pairs with high correlation coefficients.

2. Optimization

The point of optimizing portfolio weighting is to find a set of asset allocation ratios to achieve a specific investment objective. Often, this means finding the best balance between the desired rate of return and risk tolerance. Through optimization, investors can maximize returns while controlling risk, following Markowitz's Modern Portfolio Theory (MPT) and achieving efficient diversification among assets to reduce unnecessary volatility and potential losses. The optimization process considers the expected return, volatility, and correlation of assets, and uses mathematical models to find weights to achieve the optimal risk-adjusted return.
Next, we will calculate the optimal weight of a portfolio using the ten stocks from Step 1 in a different way.

Mean-variance vs. Black-Litterman model vs. Risk Parity

Mean-Variance Optimization: aims to minimize risk while maximizing expected returns by diversifying investments. The objective of optimization is to find the optimal allocation of asset weights to obtain the maximum expected return for a given level of risk, or the minimum risk for a given expected return.

Black-Litterman model: combines the features of mean-variance optimization and Capital asset pricing Model (CAPM), allowing investors to add their subjective expectations to the model to adjust the expected return on assets. It provides a structured way to integrate market equilibrium returns and the individual views of investors.

Risk Parity: The goal is to construct a portfolio in which each asset or asset class contributes equally to the overall portfolio risk. This approach typically results in relatively more investment being allocated to assets traditionally considered less risky, such as bonds, rather than riskier assets, such as stocks. A risk-balanced portfolio can theoretically provide a more stable asset allocation, especially in times of market turmoil.

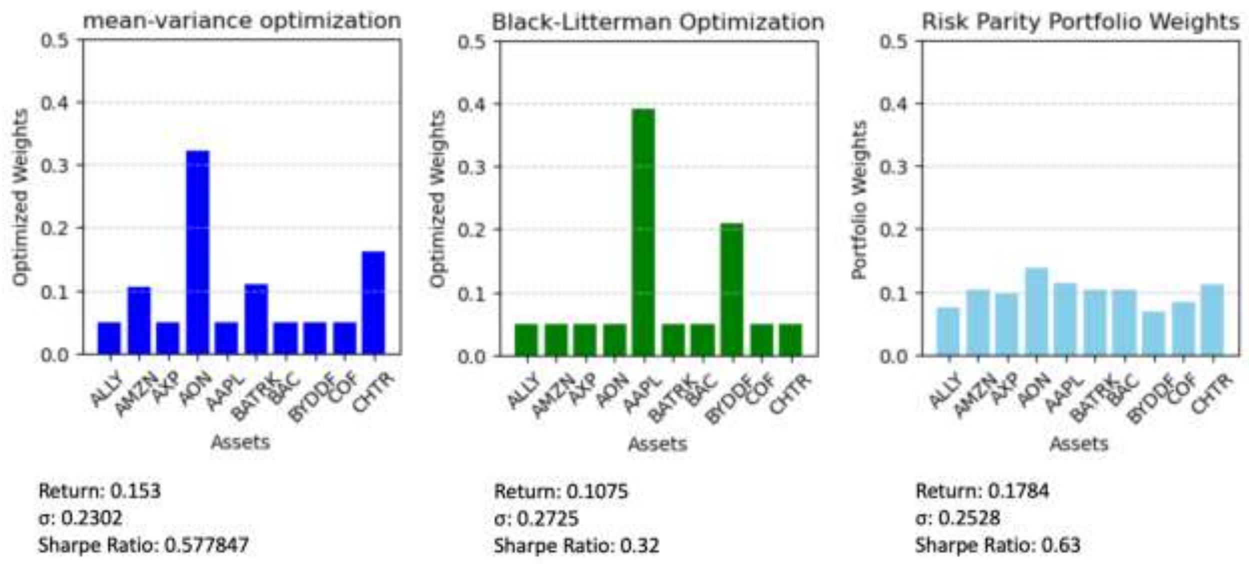


Figure 5

After the weighting optimization of this portfolio by Mean-variance, Black-Litterman and Risk Parity, we get the following results.

Mean-variance optimization shows moderate performance among the three methods, possibly due to sensitivity to expected returns. The Black-Litterman approach seems to give the most conservative portfolio in this case, sacrificing some expected returns to reduce the impact of investor judgment. The risk-parity approach results in a portfolio with a balanced rate of return and risk by distributing risk equally and has the highest Sharpe ratio of the three approaches, showing an advantage in risk-adjusted returns.

In summary, the risk parity approach provides a relatively robust portfolio, although some high-return opportunities may be missed. In contrast, mean-variance optimization may provide higher returns under certain market conditions, but it may also come with higher risks. Black-Litterman models try to find a balance between the two, considering both market expectations and investor views.

Monte Carlo

Monte Carlo simulation is a computational algorithm used to estimate and understand the impact of uncertainty on complex systems and is often used in the financial field to evaluate and optimize investment portfolios. This method forecasts the future performance of an asset portfolio through random sampling and probability statistics and provides a quantitative basis for investment decisions.

In portfolio management, Monte Carlo simulation calculates the expected return and risk of each portfolio by generating many random asset portfolio weights and then evaluates the performance of those portfolios, such as the Sharpe ratio or other risk-adjusted return measures. This approach enables investors to evaluate the performance of potential investment strategies in different market situations and select asset allocations that maximize expected returns given risk appetite.

After running 1000 simulations we got the following results：

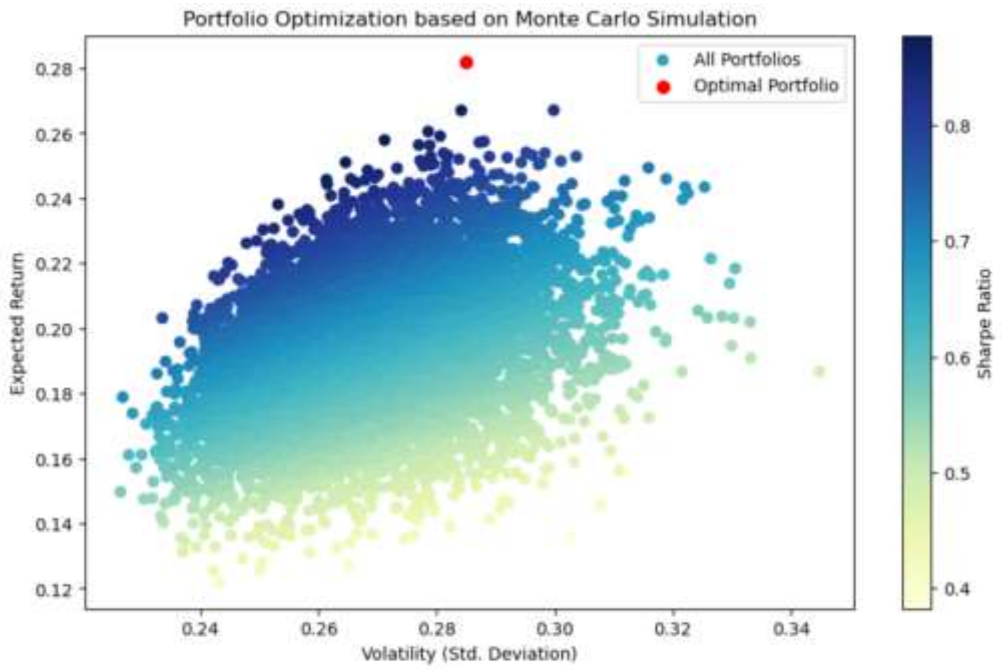


Figure 6

This graph shows the expected returns and volatility (standard deviation) of different portfolios using Monte Carlo simulations. Each dot represents a possible portfolio, with shades of color representing a high or low Sharpe ratio, light colors representing a low Sharpe ratio, and dark colors representing a high Sharpe ratio. The red dot in the graph identifies the optimal portfolio found in these simulations, that is, the portfolio with the highest Sharpe ratio, which provides high expected returns while maintaining relatively low volatility.

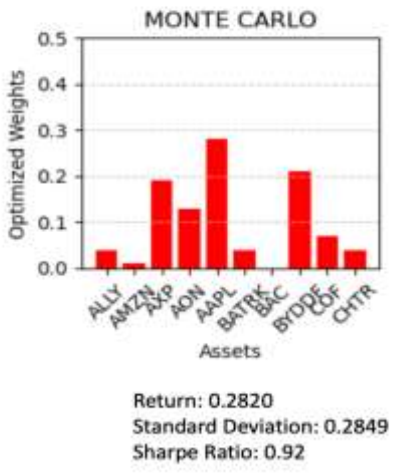


Figure 7

Through the Monte Carlo simulation of the optimal weight distribution of the portfolio, we can see that some of these assets get a high weight, such as BATRK and BYDDF, which indicates that they are particularly contributing to improving the portfolio performance during the simulation. The expected return of this portfolio is 28.20%, the standard deviation is 28.49%, and the Sharpe ratio is 0.92.

The Sharpe ratio is a measure of return per unit of risk, and a value close to 1 means that investors can expect to earn an excess return almost equal to the level of risk taken. A Sharpe ratio of 0.92 indicates a relatively high risk-adjusted return on the portfolio, i.e., the portfolio provides a good additional return per unit of total risk.

Maximum entropy portfolio maximization

Maximum entropy portfolio optimization is an advanced portfolio optimization technique that utilizes the concept of entropy to achieve the greatest possible diversification of a portfolio. Entropy here refers to the uncertainty and randomness of the weight distribution of the portfolio. By maximizing entropy, weight distribution can be derived to achieve the maximum diversity of the portfolio, thereby reducing gratuitous risk, and improving the return after adjusting risk.

Maximizing entropy means finding a distribution of asset weights such that any single asset or particular mix of assets has the most uncertain or random effect on the overall portfolio.

Monte Carlo simulation is a very useful tool when considering the best weights for a portfolio, but it can't be the only tool we use. Because Monte Carlo simulation is completely random, it can only provide approximate optimal solutions and cannot guarantee absolute optimality.

1. Define the Optimization Problem

$$H(\mathrm{w}) = -\sum_{i=1}^{n} w_i log w_i$$

2. Constraints

$$\sum_{i=1}^{n} w_i = 1;$$

$$w_i \geq 0 \text{ for all i}$$

3. Applying the Lagrange Multiplier Method

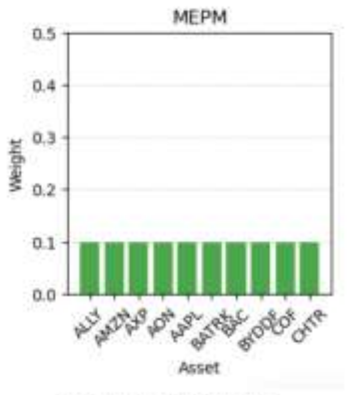
$$\mathcal{L}(\mathrm{w},\lambda) = -\sum_{i=1}^{n} w_i log w_i + \lambda(\sum_{i=1}^{n} w_i - 1)$$

$$\frac{\partial \mathcal{L}}{\partial w_i} = -\log w_i - 1 + \lambda = 0$$

$$e^{\lambda-1} = W_i$$

$$e^{\lambda-1} = \frac{1}{n} W_i = \frac{1}{n}$$

By adding constraints to the Lagrange function, the solution is that each asset is equally weighted. This shows that the optimal weight allocation that maximizes entropy, without considering any other market factors or investment constraints, is that each asset has the same weight. Such an allocation means maximizing portfolio diversity without prioritizing the expected return or risk characteristics of any asset.



Expected Return: 0.0916
Standard Deviation: 0.1509
Sharpe Ratio: 0.6067

Figure 8

The graph shows the portfolio weights obtained using the maximum entropy Portfolio Optimization method (MEPM), where each asset is assigned an equal weight. This is consistent with the results of the formula we just calculated. It seeks to maximize portfolio diversification, not favoring any particular asset, but assuming that all assets are equally important.

The expected return of the portfolio is 9.16%, the standard deviation is 15.09% and the Sharpe ratio is 0.6067. The relatively high Sharpe ratio means that the excess return per unit of risk is substantial. However, it is important to note that due to the uninformative nature of this approach, the expected rate of return and risk assessment here may not consider correlations between assets or their respective expected returns and volatility. Thus, even though this weighting method can provide maximum diversification in theory, it may not be optimal in practice, especially if the expected return or risk characteristics of some assets are significantly different from others.

Gray Wolf vs. Particle Swarm

Gray Wolf Optimizer (GWO) is a heuristic optimization algorithm that mimics the social hierarchy and hunting behavior of gray wolves. The Grey Wolf pack has a clear leadership hierarchy, and the "Wolf" in the algorithm follows the "alpha" (leader Wolf) to search for the best solution. This process involves the simulation of circling prey, tracking, and attacking, i.e. searching for the globally optimal solution in the solution space.

➓ Alpha: Alpha is the leader of the pack and determines when the pack hunts, rests, wakes up, and the direction of the hunt. In the algorithm, Alpha represents the best solution currently found.

❿ Beta Wolf: Beta Wolf is the second leader of the pack, after Alpha. If Alpha Wolf is gone, Beta Wolf will take over the leadership role. In GWO, Beta represents the second-best solution.

❿ Delta Wolf: The Delta Wolf is the third in the hierarchy and is usually responsible for the safety and discipline of the pack. In the algorithm, Delta represents the third-best solution.

Other wolves are known as Omega wolves and must obey the decisions of their superiors. In the GWO algorithm, the Alpha, Beta, and Delta Wolf solutions guide the search for Omega Wolf, while Omega Wolf explores other regions of the solution space. The whole group collaboratively searches for the global optimal solution by simulating this social hierarchical behavior. This layered approach helps the algorithm search efficiently while avoiding premature convergence to a locally optimal solution.

Particle Swarm Optimization (PSO) is another swarm-based optimization algorithm inspired by the social behavior of flocks of birds or schools of fish. In PSO, each "particle" represents a potential solution in the solution space and updates its position by tracking the best solution in individual and group experiences. The particle adjusts its flight path according to its own experience and that of the group.

Velocity Update Formula:

$$v[]=w\times v[]+c_1\times r_1\times(pbest[]-present[])+c_2\times r_2\times(gbest[]-present[])$$

w is the inertia weight
$c1$ and $c2$ are learning factors
$r1$ and $r2$ are random numbers
pbest[] is the particle's historical best position
gbest[] is the global best position
Position Update Formula:

$$position[]=position[]+v[]$$

GWO and PSO are both swarm intelligence optimization algorithms that mimic the social behavior of biological groups to solve optimization problems, but the biggest difference between them is that they mimic natural behavior and update mechanisms. GWO mimics the grey Wolf's social hierarchy and cooperative hunting strategy. Individuals in the algorithm update their position based on the position of the top three individuals in the group (Alpha, Beta, and Delta). PSO simulates the movement of flocks of birds or schools of fish. Each particle updates its speed and position based on its own experience (the individual optimal solution) and the group's experience (the global optimal solution). In PSO, all particles are equally important and there is no hierarchical social structure.

After using GWO and PSO respectively, we obtained the following results.

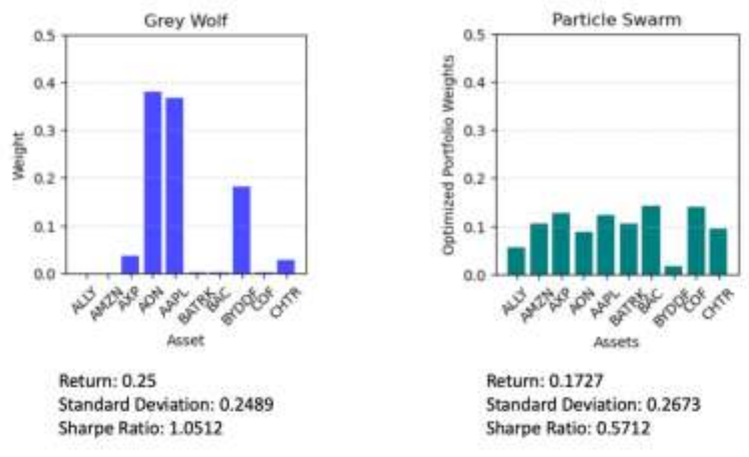


Figure 9

Gray Wolf's weight distribution has one or several assets with significantly higher weights than others, while Particle Swarm: the weights are more evenly distributed, suggesting that the algorithm may place more emphasis on diversification across the portfolio.

Gray Wolf: A high return (25%), a relatively low standard deviation (24.89%), and a Sharpe ratio of 1.0512, which indicates a relatively high risk-adjusted return for the portfolio. Particle Swarm: A return of 17.27%, a standard deviation of 26.73%, and a Sharpe ratio of 0.5712, which means that PSO has a lower risk-adjusted return on the portfolio relative to Gray Wolf optimization.

In summary, the Gray Wolf algorithm results in a more concentrated portfolio allocation in this example and exhibits a higher risk-adjusted return (higher Sharpe ratio). The Particle Swarm algorithm, on the other hand, favors a more decentralized allocation of assets but is slightly inferior in performance metrics.

3. Forecasting

Forecasting plays a key role in financial investment decisions, especially in predicting the return on assets. Accurate forecasts can help investors develop strategies, assess risks, and optimize portfolios. We will introduce ARIMA (autoregressive Integral Moving Average Model) and LSTM (Long Short-Term Memory Model), two popular time series analysis methods used to predict the future direction of financial markets. Both provide powerful tools for the prediction of investment rate of return, which can help investors find possible trends and patterns in a market full of uncertainty, to make more informed investment choices.

Autoregressive Integral Moving Average Model

ARIMA (Autoregressive Integrated Moving Average Model) is one of the most used models for time series forecasting. The ARIMA model combines three main time series modeling concepts: autoregression (AR), difference (I), and moving average (MA). The ARIMA model is usually expressed as ARIMA(p,d,q)

Autoregressive (AR):
- The autoregressive part considers the effect of observations at previous time points on current observations.
- p is the order of the AR term, representing the number of time steps to be retraced.

Difference (I):
- Difference is meant to make time series data stationary, even if its mean and variance do not change over time.
- d is the degree of difference to make the sequence stationary.

Moving Average (MA):
- The moving average section considers the effect of the error item in the time series and its lag value on the observed value at the current point in time.
- q is the order of the MA term, i.e., the number of error terms in the previous period.
-

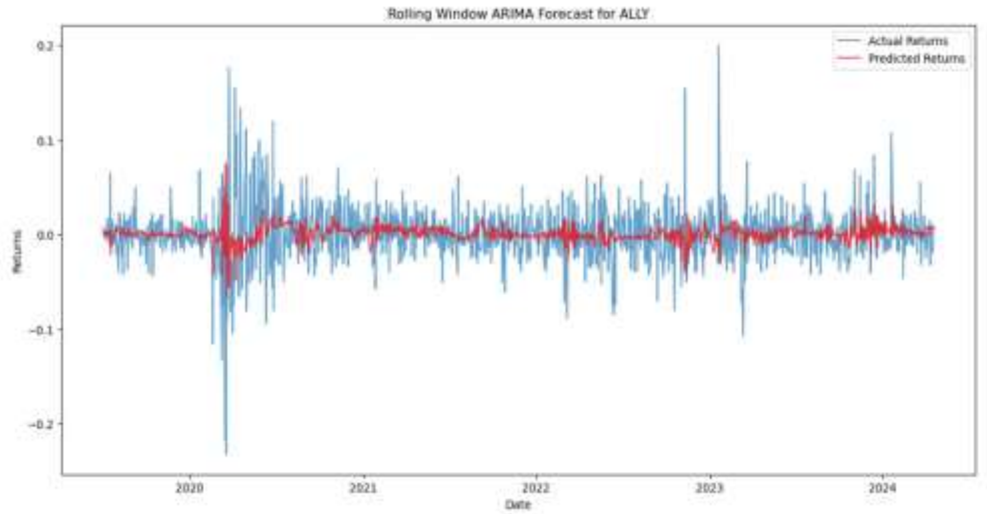


Figure 10

This graph shows the results of rolling window prediction using the ARIMA model. The blue line represents the actual stock return, while the red line represents the return predicted by the

ARIMA model. Over many periods, the model's predictions appear to match actual returns, suggesting that ARIMA captures some basic patterns in equity returns.

However, models are poor at capturing extreme values, especially during periods of large swings in returns. These extreme movements can be caused by market-specific events or macroeconomic factors that are often difficult to predict entirely from historical price data.

The gap between model predictions and actual data indicates that forecast accuracy can be affected by multiple market forces. Nevertheless, as a time series analysis tool, the ARIMA model can still provide useful information about future stock price trends, especially under more stable market conditions. For investors, combining other types of analysis, such as fundamental analysis or other technical indicators, may improve the accuracy of forecasts.

Long Short-Term Memory Model

LSTM (Long Short-Term Memory) is a special type of recurrent neural network (RNN) that is capable of learning and remembering long-term dependent information. It avoids the gradient disappearance or explosion problems faced by traditional RNNS through a gate control mechanism, which makes LSTMS very efficient when dealing with long sequences of data.

Structure of an LSTM Unit:

Forget Gate:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f \right)$$

Input Gate:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i \right)$$

$$\widetilde{C_t} = tant(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Cell State Update:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}$$

Output Gate:

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_0 \right)$$

$$h_t = o_t * \tanh(C_t)$$

The LSTM unit processes data through several key steps as well as a series of gate control mechanisms, maintaining and updating its state in the process. These gate control mechanisms allow LSTM units to remember important information and forget irrelevant information.

- Forget Gate: Determines what information should be discarded from the state of the cell. Where σ is the sigmoid function, it outputs a value between 0 and 1, where 0 means completely forgotten and 1 means completely retained.
- Input Gate: Determines what new information will be stored in the cell state. It consists of two parts: a sigmoid layer that decides which values we will update, and a tanh layer that generates candidate value vectors.
- Cell State Update: Then update the cell status by combining the information of the forgotten gate and the input gate.
- Output Gate: Finally, the output gate determines the next hidden state.

The Sigmoid function limits the output between 0 and 1, allowing the network to make binary choices, while the tanh function outputs values between -1 and 1, helping to regulate the intensity of the information flow. The combination of these nonlinear functions allows LSTM to handle complex data patterns while being more efficient at preserving long-term dependencies.

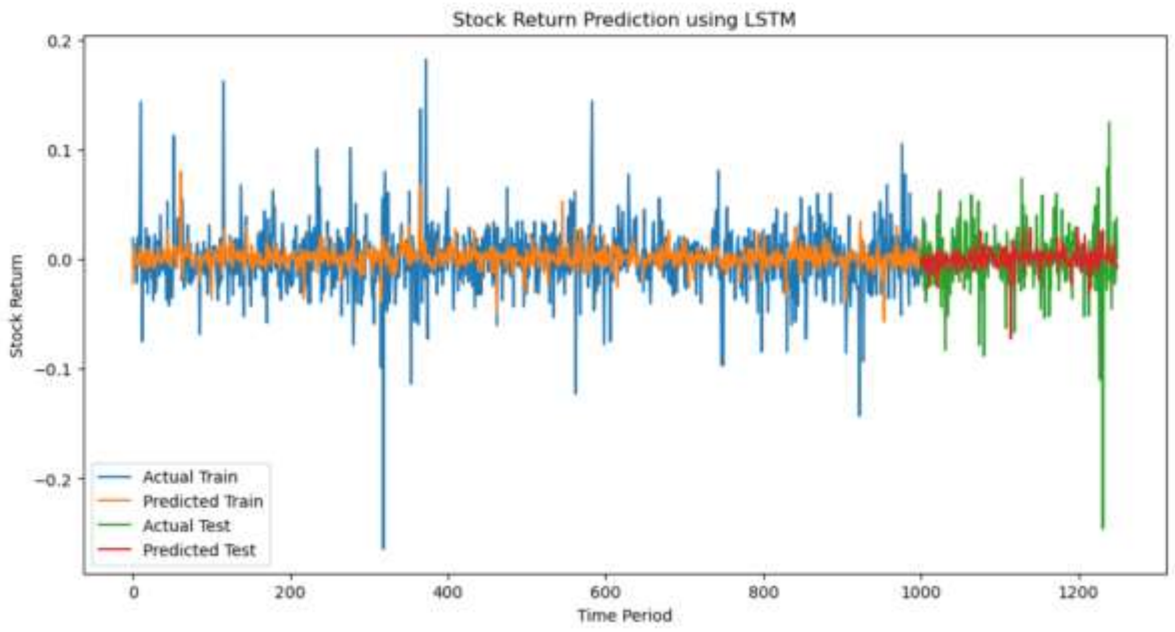


Figure 11

This graph shows the results of using the Long short-term Memory Network (LSTM) model to predict stock returns. The blue line in the graph represents the actual stock return in the training set, and the orange line is the model's prediction during the training. On the test set, the green line represents the actual stock return, while the red line represents the predicted return of the model. It can be observed from the figure that the model fits the training data quite closely, and the predicted curve is generally consistent with the actual data curve. On the test set, the model captures the actual trend, although at some points in time, there is a deviation between the prediction and the actual value. Such models are particularly good at capturing long-term dependencies on time series data, which is particularly important in financial market forecasting because the historical behavior of markets often has an impact on future price movements. The application of the LSTM model helps investors and analysts to better understand market dynamics and find value in complex financial data.

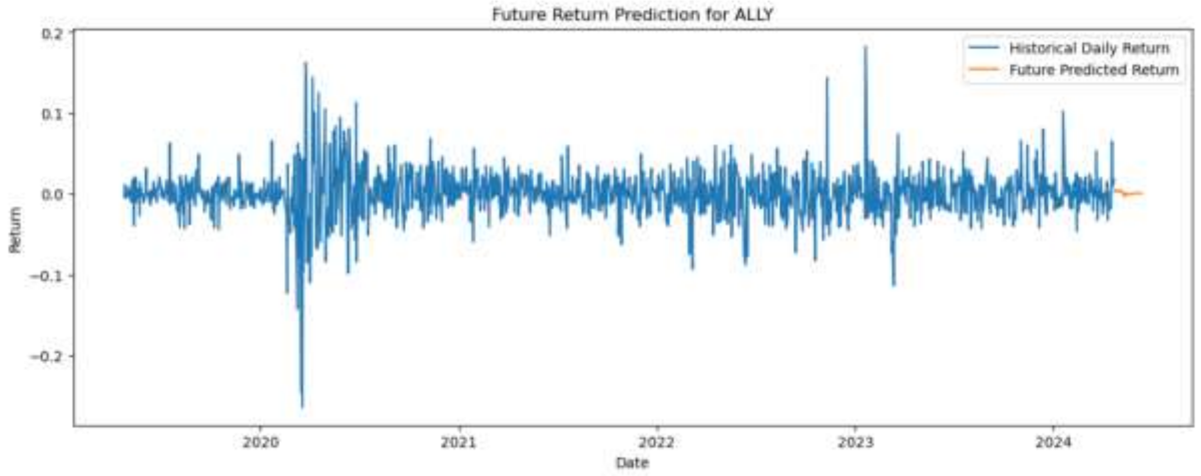


Figure 12

In this chart, we use the LSTM model to predict the future return of stock ALLY. Historical data (the blue line) is used to train the model, while the forecasts produced by the model (the orange line) depict estimates of future market conditions. The prediction section is immediately at the end of the historical data, indicating that the model is trained on the most recent historical data and used to generate predictions of future values. The predictions generated by the model are a continuation of historical trends and reflect patterns of historical behavior.

ARIMA vs. LSTM

ARIMA and LSTM are two common time series prediction models, which differ in their ability to process data and the scenarios they apply.

ARIMA is a statistical model for linear and stationary time series data. It relies on the autoregressive (AR) part of the data, the difference (I) to achieve stationarity, and the moving average (MA) part to capture short-term correlations and seasonal patterns in the time series. The main advantages of ARIMA are that the model is simple and efficient, and the parameters are easy to understand and identify, making it ideal for short-term forecasting. However, it is limited in that it cannot capture long-term dependencies and is limited to dealing with linear relationships.

In contrast, LSTM, as a deep learning model, is designed to process serial data with long-term dependence and nonlinear characteristics. Through its complex network structure, especially gate-controlled memory cells, LSTM can learn long-term patterns in time series, even when the data is non-stationary. LSTM excels at capturing and predicting complex patterns, such as those commonly found in stock prices or financial market forecasts. However, the disadvantages of such models include being computationally more intensive, requiring large amounts of data for training, and often more complex in tuning the parameters of the models.

When choosing the right model, you need to consider the characteristics of the data and the target of the prediction. ARIMA is better suited for problems with clear seasonality and short-term correlations, while LSTM is best suited for scenarios where long-term dependencies and complex patterns need to be captured. In practical applications, the two can be selected or combined according to specific needs and data characteristics.

Conclusion

In the financial field, there are various methods to seek the optimal portfolio weight and forecast the return on assets, and each method has its unique advantages and applicable scenarios based on its theoretical basis and assumptions.

Mean-variance optimization focuses on the balance of risk and return; Risk parity seeks to spread risk; Black-Litterman model combines market equilibrium and subjective view; Monte Carlo simulation explores potential possibilities through random processes; The maximum entropy method seeks the maximum diversity; Nature-inspired optimization algorithms, such as Gray Wolf and Particle Swarm, try to find solutions through strategies that mimic biological behavior. When it comes to predicting returns, ARIMA and LSTM represent applications of traditional statistics and modern deep learning, respectively, with each model providing insights into the data features in which it excels.

In summary, there is no one "best" model or method, only the "best fit" choice. Choosing the most suitable instrument depends on specific investment objectives, market conditions, data characteristics, and risk appetite. An investor or analyst with a deep understanding of these methods has the flexibility to select and combine these tools to develop the best investment strategy based on different situations and objectives. Therefore, understanding the principles and limitations of each approach is key to effective financial analysis and decision-making.

Literature

grey wolf optimization - https://www.sciencedirect.com/science/article/pii/S0965997813001853

https://www.geeksforgeeks.org/grey-wolf-optimization-introduction/

https://seyedalimirjalili.com/gwo

transaction cost analysis - https://sigtech.com/insights/transaction-cost-analysis/

https://www.researchgate.net/publication/373896087_Application_of_ARIMA_Model_in_Portfolio_Optimization

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9759213

lstm - https://ieeexplore.ieee.org/document/9641662

https://www.sciencedirect.com/science/article/pii/S2193943821001175

https://arxiv.org/abs/2111.04709

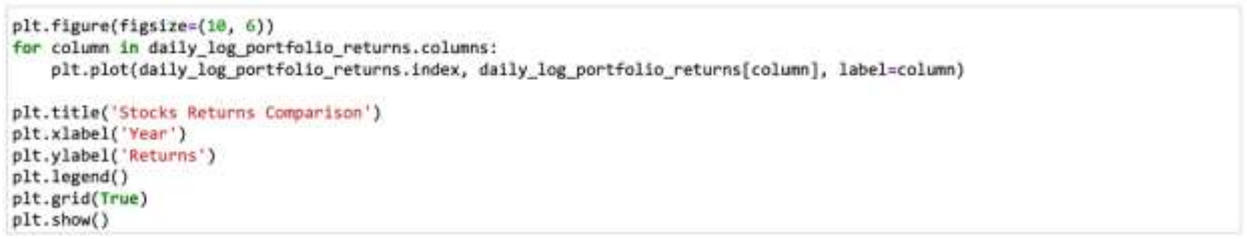
https://cse.hkust.edu.hk/~rossiter/independent_studies_projects/lstm_portfolio_optimization/lstm_portfolio_optimization.pdf

https://www.researchgate.net/publication/376131755_Portfolio_Optimization_Based_on_the_LSTM_Forecasting_Model

Appendix

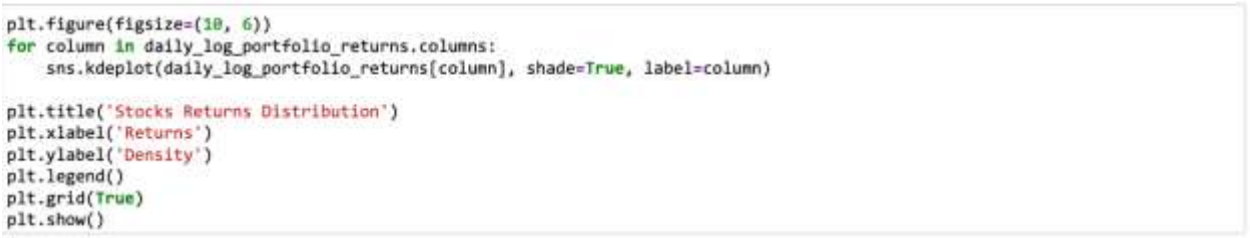
Code

Log return



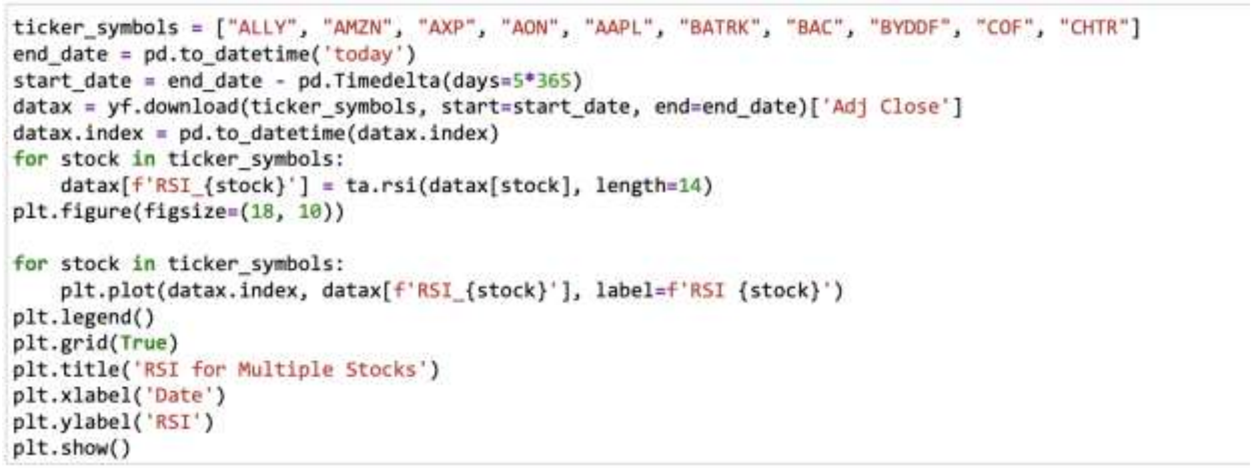
```python
plt.figure(figsize=(10, 6))
for column in daily_log_portfolio_returns.columns:
    plt.plot(daily_log_portfolio_returns.index, daily_log_portfolio_returns[column], label=column)

plt.title('Stocks Returns Comparison')
plt.xlabel('Year')
plt.ylabel('Returns')
plt.legend()
plt.grid(True)
plt.show()
```

Distribution of return



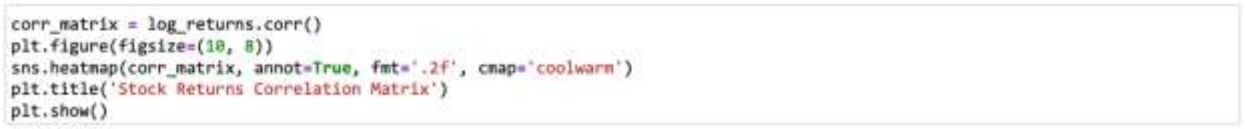
```
plt.figure(figsize=(10, 6))
for column in daily_log_portfolio_returns.columns:
    sns.kdeplot(daily_log_portfolio_returns[column], shade=True, label=column)

plt.title('Stocks Returns Distribution')
plt.xlabel('Returns')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```

RSI



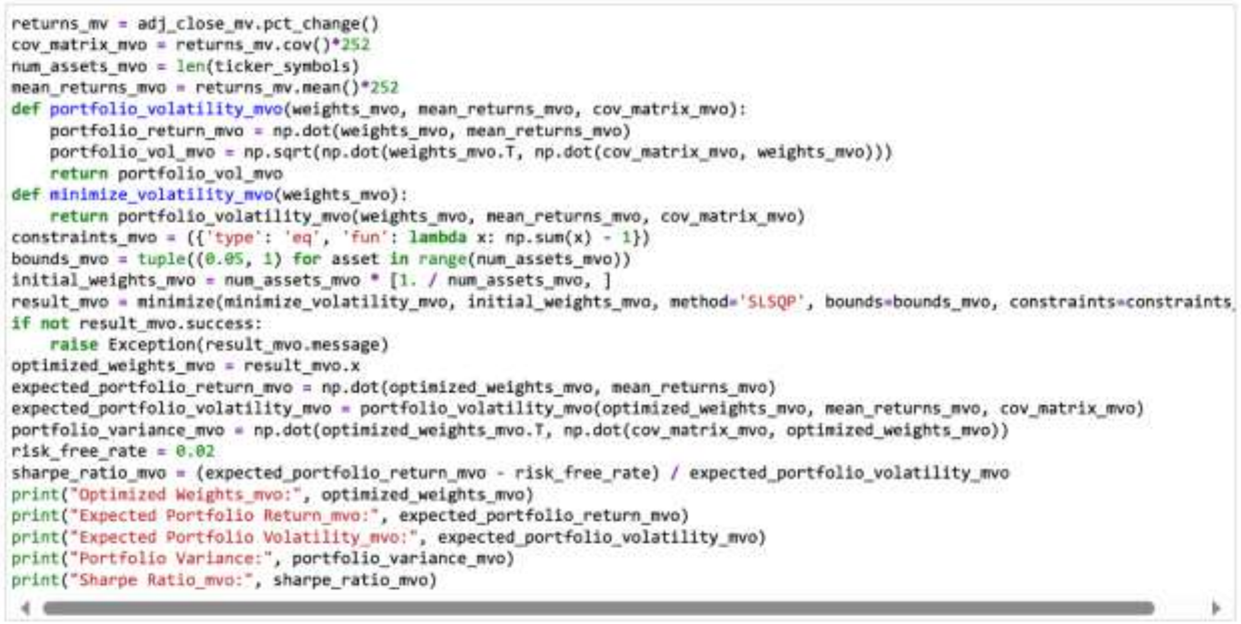
```
ticker_symbols = ["ALLY", "AMZN", "AXP", "AON", "AAPL", "BATRK", "BAC", "BYDDF", "COF", "CHTR"]
end_date = pd.to_datetime('today')
start_date = end_date - pd.Timedelta(days=5*365)
datax = yf.download(ticker_symbols, start=start_date, end=end_date)['Adj Close']
datax.index = pd.to_datetime(datax.index)
for stock in ticker_symbols:
    datax[f'RSI_{stock}'] = ta.rsi(datax[stock], length=14)
plt.figure(figsize=(18, 10))

for stock in ticker_symbols:
    plt.plot(datax.index, datax[f'RSI_{stock}'], label=f'RSI {stock}')
plt.legend()
plt.grid(True)
plt.title('RSI for Multiple Stocks')
plt.xlabel('Date')
plt.ylabel('RSI')
plt.show()
```

Correlation matrix



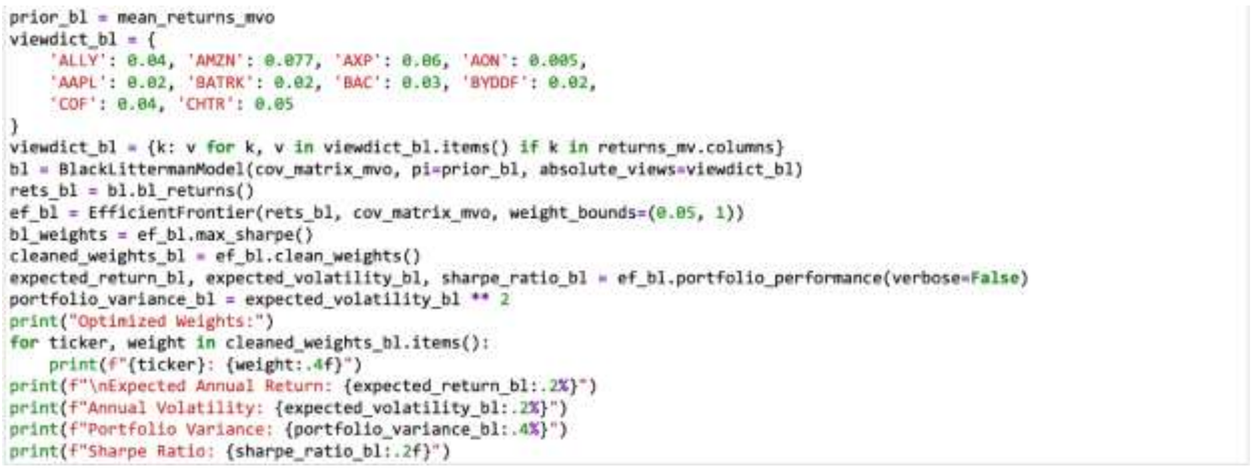
```
corr_matrix = log_returns.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Stock Returns Correlation Matrix')
plt.show()
```

Mean-variance optimization



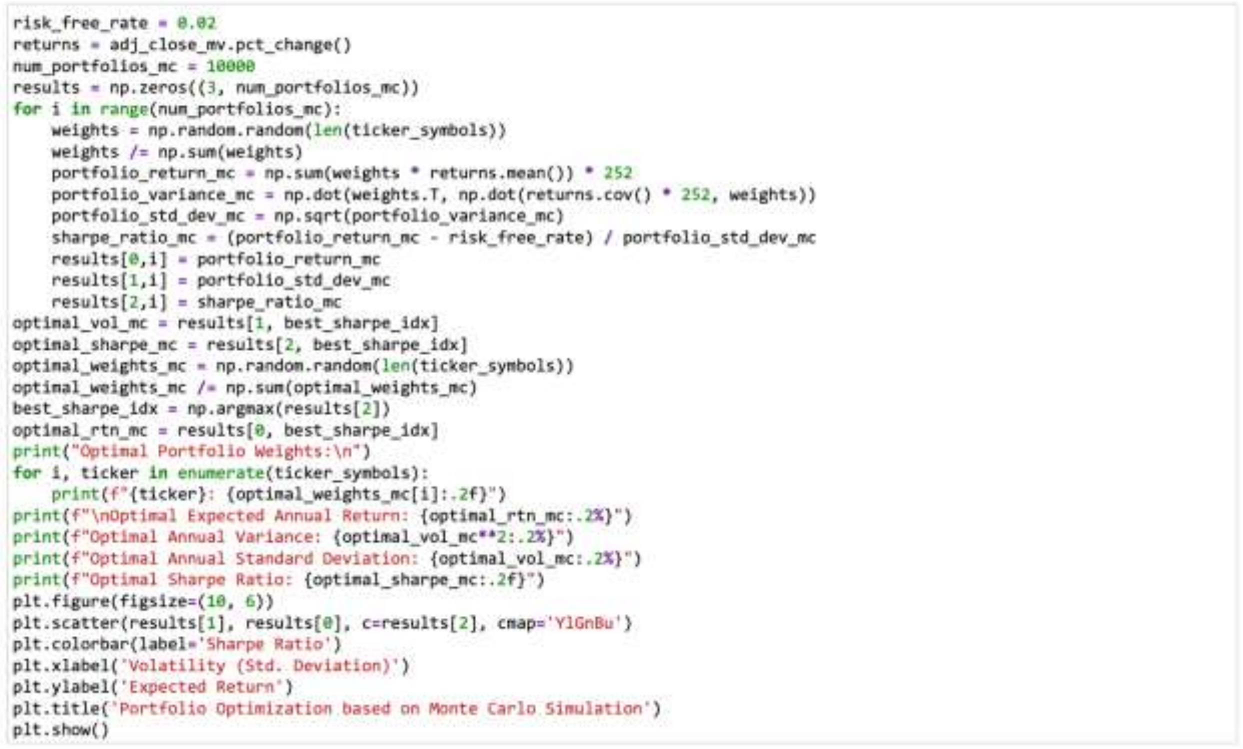
```
returns_mv = adj_close_mv.pct_change()
cov_matrix_mvo = returns_mv.cov()*252
num_assets_mvo = len(ticker_symbols)
mean_returns_mvo = returns_mv.mean()*252
def portfolio_volatility_mvo(weights_mvo, mean_returns_mvo, cov_matrix_mvo):
    portfolio_return_mvo = np.dot(weights_mvo, mean_returns_mvo)
    portfolio_vol_mvo = np.sqrt(np.dot(weights_mvo.T, np.dot(cov_matrix_mvo, weights_mvo)))
    return portfolio_vol_mvo
def minimize_volatility_mvo(weights_mvo):
    return portfolio_volatility_mvo(weights_mvo, mean_returns_mvo, cov_matrix_mvo)
constraints_mvo = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bounds_mvo = tuple((0.05, 1) for asset in range(num_assets_mvo))
initial_weights_mvo = num_assets_mvo * [1. / num_assets_mvo, ]
result_mvo = minimize(minimize_volatility_mvo, initial_weights_mvo, method='SLSQP', bounds=bounds_mvo, constraints=constraints_
if not result_mvo.success:
    raise Exception(result_mvo.message)
optimized_weights_mvo = result_mvo.x
expected_portfolio_return_mvo = np.dot(optimized_weights_mvo, mean_returns_mvo)
expected_portfolio_volatility_mvo = portfolio_volatility_mvo(optimized_weights_mvo, mean_returns_mvo, cov_matrix_mvo)
portfolio_variance_mvo = np.dot(optimized_weights_mvo.T, np.dot(cov_matrix_mvo, optimized_weights_mvo))
risk_free_rate = 0.02
sharpe_ratio_mvo = (expected_portfolio_return_mvo - risk_free_rate) / expected_portfolio_volatility_mvo
print("Optimized Weights_mvo:", optimized_weights_mvo)
print("Expected Portfolio Return_mvo:", expected_portfolio_return_mvo)
print("Expected Portfolio Volatility_mvo:", expected_portfolio_volatility_mvo)
print("Portfolio Variance:", portfolio_variance_mvo)
print("Sharpe Ratio_mvo:", sharpe_ratio_mvo)
```

Black-Litterman



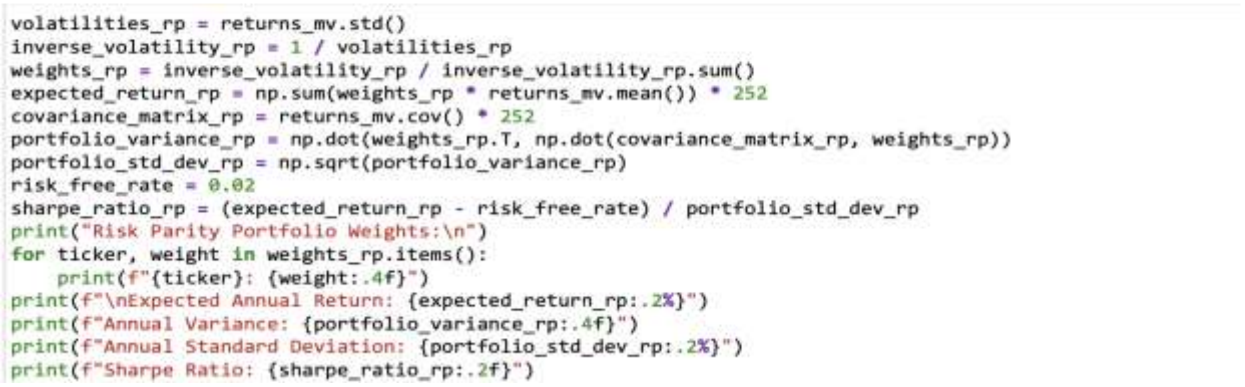
```
prior_bl = mean_returns_mvo
viewdict_bl = {
    'ALLY': 0.04, 'AMZN': 0.077, 'AXP': 0.06, 'AON': 0.005,
    'AAPL': 0.02, 'BATRK': 0.02, 'BAC': 0.03, 'BYDDF': 0.02,
    'COF': 0.04, 'CHTR': 0.05
}
viewdict_bl = {k: v for k, v in viewdict_bl.items() if k in returns_mv.columns}
bl = BlackLittermanModel(cov_matrix_mvo, pi=prior_bl, absolute_views=viewdict_bl)
rets_bl = bl.bl_returns()
ef_bl = EfficientFrontier(rets_bl, cov_matrix_mvo, weight_bounds=(0.05, 1))
bl_weights = ef_bl.max_sharpe()
cleaned_weights_bl = ef_bl.clean_weights()
expected_return_bl, expected_volatility_bl, sharpe_ratio_bl = ef_bl.portfolio_performance(verbose=False)
portfolio_variance_bl = expected_volatility_bl ** 2
print("Optimized Weights:")
for ticker, weight in cleaned_weights_bl.items():
    print(f"{ticker}: {weight:.4f}")
print(f"\nExpected Annual Return: {expected_return_bl:.2%}")
print(f"Annual Volatility: {expected_volatility_bl:.2%}")
print(f"Portfolio Variance: {portfolio_variance_bl:.4%}")
print(f"Sharpe Ratio: {sharpe_ratio_bl:.2f}")
```

MONTE CARLO



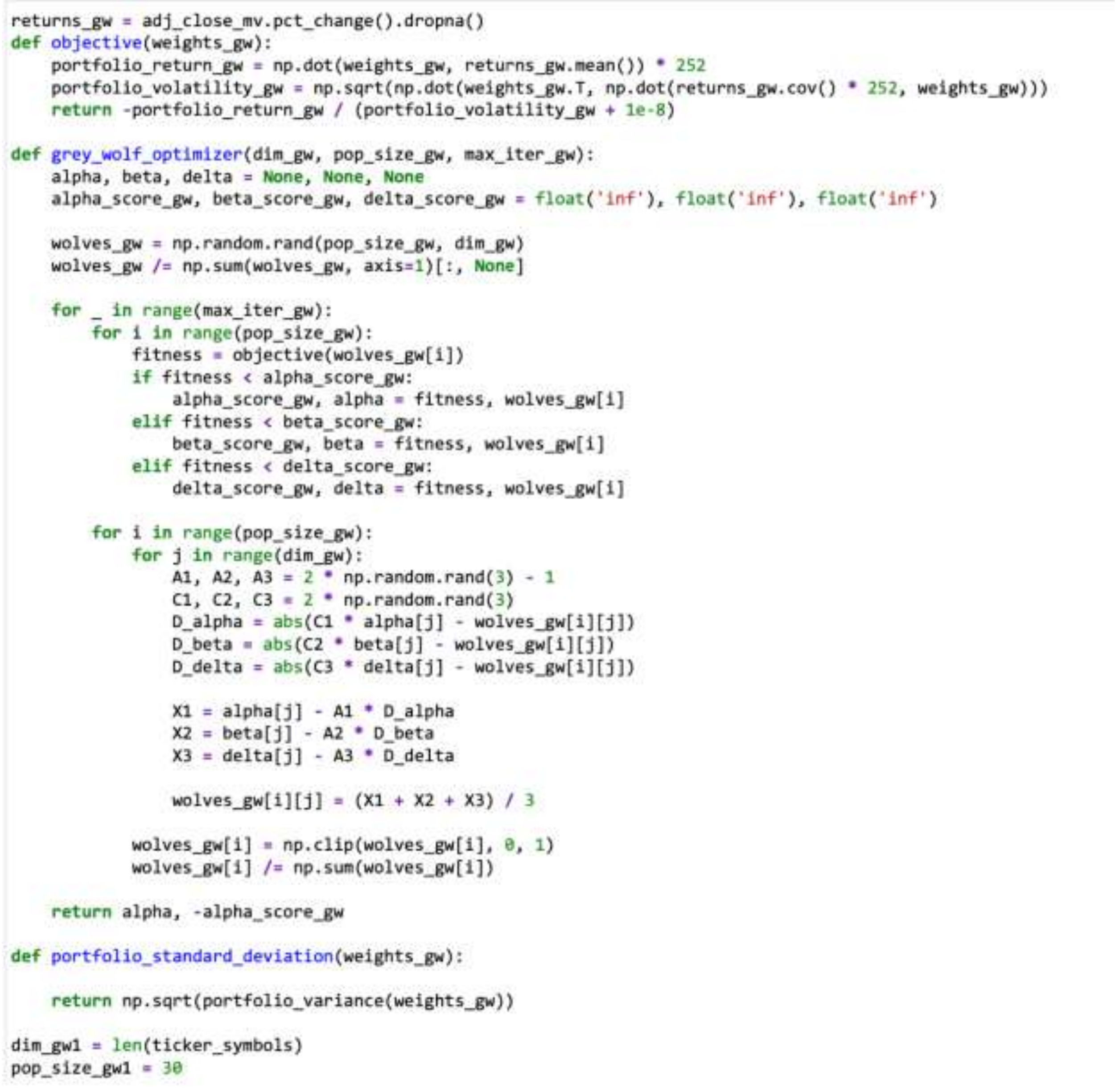
```
risk_free_rate = 0.02
returns = adj_close_mv.pct_change()
num_portfolios_mc = 10000
results = np.zeros((3, num_portfolios_mc))
for i in range(num_portfolios_mc):
    weights = np.random.random(len(ticker_symbols))
    weights /= np.sum(weights)
    portfolio_return_mc = np.sum(weights * returns.mean()) * 252
    portfolio_variance_mc = np.dot(weights.T, np.dot(returns.cov() * 252, weights))
    portfolio_std_dev_mc = np.sqrt(portfolio_variance_mc)
    sharpe_ratio_mc = (portfolio_return_mc - risk_free_rate) / portfolio_std_dev_mc
    results[0,i] = portfolio_return_mc
    results[1,i] = portfolio_std_dev_mc
    results[2,i] = sharpe_ratio_mc
optimal_vol_mc = results[1, best_sharpe_idx]
optimal_sharpe_mc = results[2, best_sharpe_idx]
optimal_weights_mc = np.random.random(len(ticker_symbols))
optimal_weights_mc /= np.sum(optimal_weights_mc)
best_sharpe_idx = np.argmax(results[2])
optimal_rtn_mc = results[0, best_sharpe_idx]
print("Optimal Portfolio Weights:\n")
for i, ticker in enumerate(ticker_symbols):
    print(f"{ticker}: {optimal_weights_mc[i]:.2f}")
print(f"\nOptimal Expected Annual Return: {optimal_rtn_mc:.2%}")
print(f"Optimal Annual Variance: {optimal_vol_mc**2:.2%}")
print(f"Optimal Annual Standard Deviation: {optimal_vol_mc:.2%}")
print(f"Optimal Sharpe Ratio: {optimal_sharpe_mc:.2f}")
plt.figure(figsize=(10, 6))
plt.scatter(results[1], results[0], c=results[2], cmap='YlGnBu')
plt.colorbar(label='Sharpe Ratio')
plt.xlabel('Volatility (Std. Deviation)')
plt.ylabel('Expected Return')
plt.title('Portfolio Optimization based on Monte Carlo Simulation')
plt.show()
```

RISK PARITY

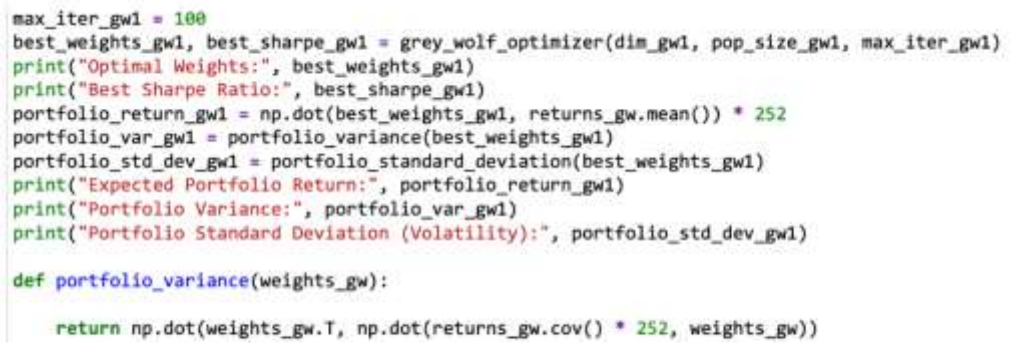


```
volatilities_rp = returns_mv.std()
inverse_volatility_rp = 1 / volatilities_rp
weights_rp = inverse_volatility_rp / inverse_volatility_rp.sum()
expected_return_rp = np.sum(weights_rp * returns_mv.mean()) * 252
covariance_matrix_rp = returns_mv.cov() * 252
portfolio_variance_rp = np.dot(weights_rp.T, np.dot(covariance_matrix_rp, weights_rp))
portfolio_std_dev_rp = np.sqrt(portfolio_variance_rp)
risk_free_rate = 0.02
sharpe_ratio_rp = (expected_return_rp - risk_free_rate) / portfolio_std_dev_rp
print("Risk Parity Portfolio Weights:\n")
for ticker, weight in weights_rp.items():
    print(f"{ticker}: {weight:.4f}")
print(f"\nExpected Annual Return: {expected_return_rp:.2%}")
print(f"Annual Variance: {portfolio_variance_rp:.4f}")
print(f"Annual Standard Deviation: {portfolio_std_dev_rp:.2%}")
print(f"Sharpe Ratio: {sharpe_ratio_rp:.2f}")
```

grey wolf optimization

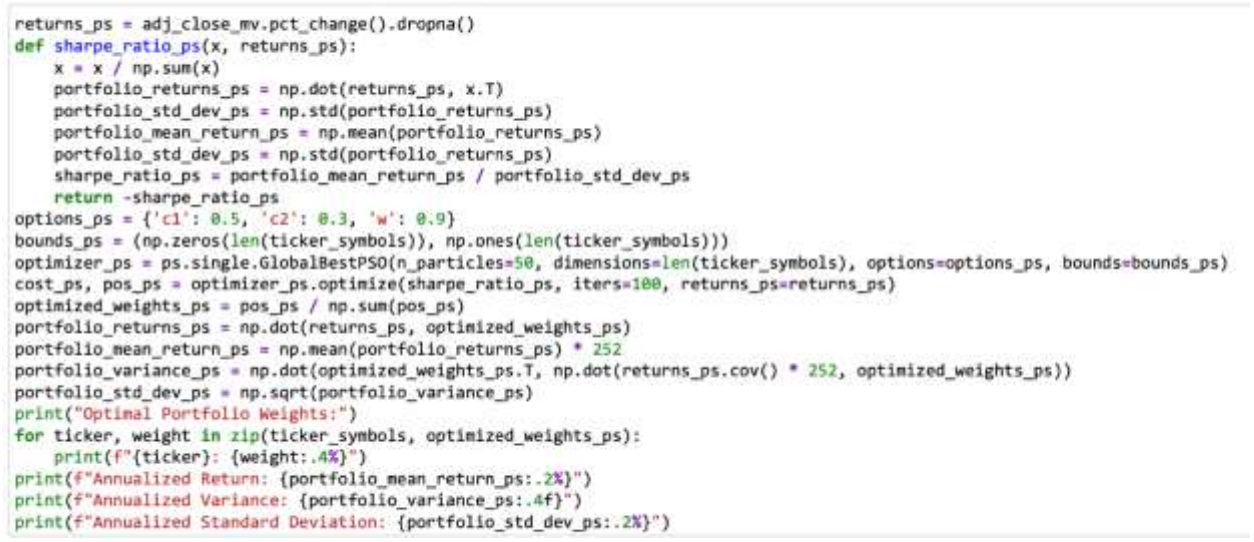


```
returns_gw = adj_close_mv.pct_change().dropna()
def objective(weights_gw):
    portfolio_return_gw = np.dot(weights_gw, returns_gw.mean()) * 252
    portfolio_volatility_gw = np.sqrt(np.dot(weights_gw.T, np.dot(returns_gw.cov() * 252, weights_gw)))
    return -portfolio_return_gw / (portfolio_volatility_gw + 1e-8)

def grey_wolf_optimizer(dim_gw, pop_size_gw, max_iter_gw):
    alpha, beta, delta = None, None, None
    alpha_score_gw, beta_score_gw, delta_score_gw = float('inf'), float('inf'), float('inf')

    wolves_gw = np.random.rand(pop_size_gw, dim_gw)
    wolves_gw /= np.sum(wolves_gw, axis=1)[:, None]

    for _ in range(max_iter_gw):
        for i in range(pop_size_gw):
            fitness = objective(wolves_gw[i])
            if fitness < alpha_score_gw:
                alpha_score_gw, alpha = fitness, wolves_gw[i]
            elif fitness < beta_score_gw:
                beta_score_gw, beta = fitness, wolves_gw[i]
            elif fitness < delta_score_gw:
                delta_score_gw, delta = fitness, wolves_gw[i]

        for i in range(pop_size_gw):
            for j in range(dim_gw):
                A1, A2, A3 = 2 * np.random.rand(3) - 1
                C1, C2, C3 = 2 * np.random.rand(3)
                D_alpha = abs(C1 * alpha[j] - wolves_gw[i][j])
                D_beta = abs(C2 * beta[j] - wolves_gw[i][j])
                D_delta = abs(C3 * delta[j] - wolves_gw[i][j])

                X1 = alpha[j] - A1 * D_alpha
                X2 = beta[j] - A2 * D_beta
                X3 = delta[j] - A3 * D_delta

                wolves_gw[i][j] = (X1 + X2 + X3) / 3

            wolves_gw[i] = np.clip(wolves_gw[i], 0, 1)
            wolves_gw[i] /= np.sum(wolves_gw[i])

    return alpha, -alpha_score_gw

def portfolio_standard_deviation(weights_gw):

    return np.sqrt(portfolio_variance(weights_gw))

dim_gw1 = len(ticker_symbols)
pop_size_gw1 = 30
```



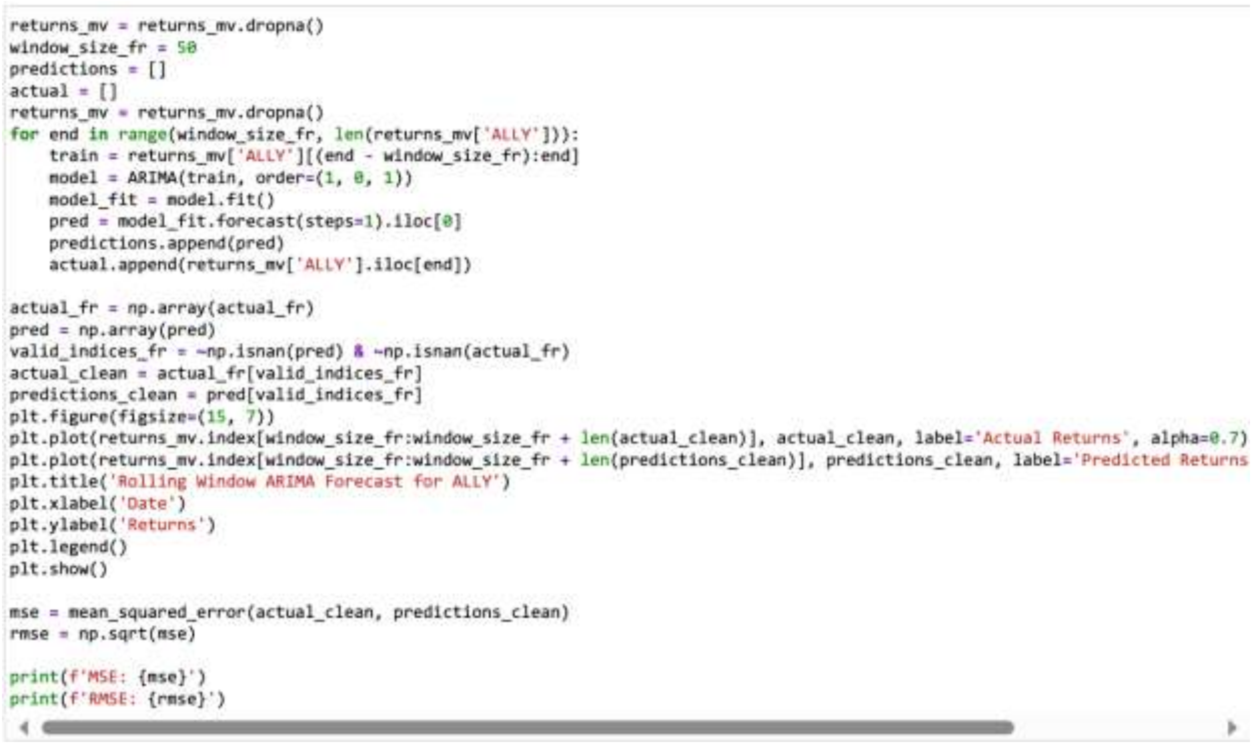
```
max_iter_gw1 = 100
best_weights_gw1, best_sharpe_gw1 = grey_wolf_optimizer(dim_gw1, pop_size_gw1, max_iter_gw1)
print("Optimal Weights:", best_weights_gw1)
print("Best Sharpe Ratio:", best_sharpe_gw1)
portfolio_return_gw1 = np.dot(best_weights_gw1, returns_gw.mean()) * 252
portfolio_var_gw1 = portfolio_variance(best_weights_gw1)
portfolio_std_dev_gw1 = portfolio_standard_deviation(best_weights_gw1)
print("Expected Portfolio Return:", portfolio_return_gw1)
print("Portfolio Variance:", portfolio_var_gw1)
print("Portfolio Standard Deviation (Volatility):", portfolio_std_dev_gw1)

def portfolio_variance(weights_gw):

    return np.dot(weights_gw.T, np.dot(returns_gw.cov() * 252, weights_gw))
```

PARTICLE SWARN



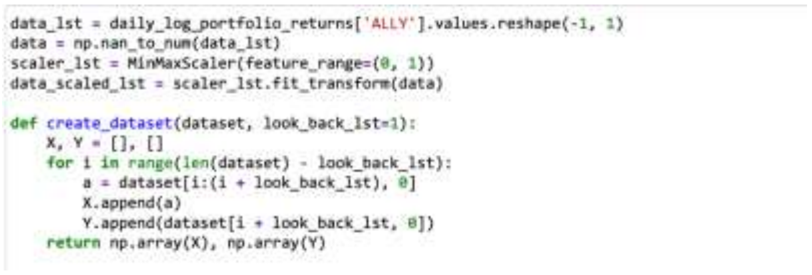
```
returns_ps = adj_close_mv.pct_change().dropna()
def sharpe_ratio_ps(x, returns_ps):
    x = x / np.sum(x)
    portfolio_returns_ps = np.dot(returns_ps, x.T)
    portfolio_std_dev_ps = np.std(portfolio_returns_ps)
    portfolio_mean_return_ps = np.mean(portfolio_returns_ps)
    portfolio_std_dev_ps = np.std(portfolio_returns_ps)
    sharpe_ratio_ps = portfolio_mean_return_ps / portfolio_std_dev_ps
    return -sharpe_ratio_ps
options_ps = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
bounds_ps = (np.zeros(len(ticker_symbols)), np.ones(len(ticker_symbols)))
optimizer_ps = ps.single.GlobalBestPSO(n_particles=50, dimensions=len(ticker_symbols), options=options_ps, bounds=bounds_ps)
cost_ps, pos_ps = optimizer_ps.optimize(sharpe_ratio_ps, iters=100, returns_ps=returns_ps)
optimized_weights_ps = pos_ps / np.sum(pos_ps)
portfolio_returns_ps = np.dot(returns_ps, optimized_weights_ps)
portfolio_mean_return_ps = np.mean(portfolio_returns_ps) * 252
portfolio_variance_ps = np.dot(optimized_weights_ps.T, np.dot(returns_ps.cov() * 252, optimized_weights_ps))
portfolio_std_dev_ps = np.sqrt(portfolio_variance_ps)
print("Optimal Portfolio Weights:")
for ticker, weight in zip(ticker_symbols, optimized_weights_ps):
    print(f"{ticker}: {weight:.4%}")
print(f"Annualized Return: {portfolio_mean_return_ps:.2%}")
print(f"Annualized Variance: {portfolio_variance_ps:.4f}")
print(f"Annualized Standard Deviation: {portfolio_std_dev_ps:.2%}")
```

ARIMA

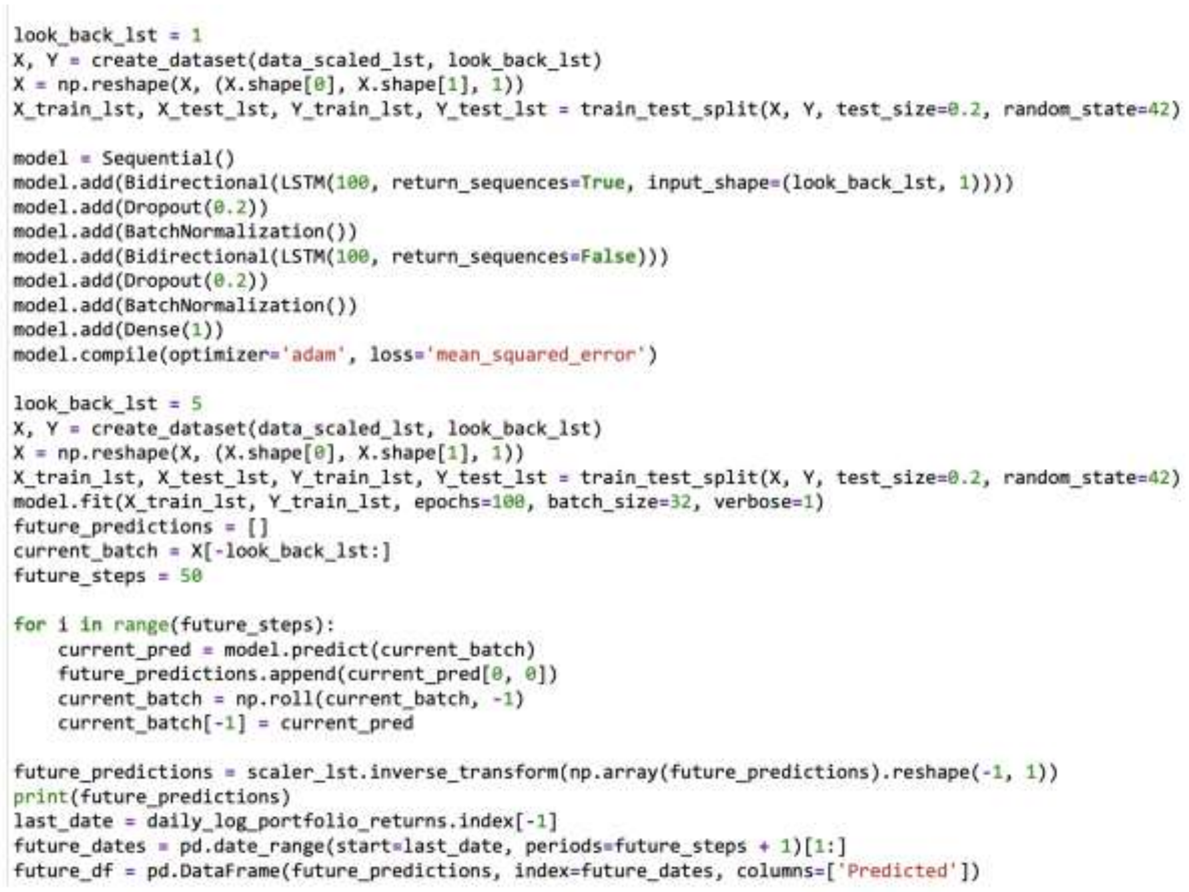


```
returns_mv = returns_mv.dropna()
window_size_fr = 50
predictions = []
actual = []
returns_mv = returns_mv.dropna()
for end in range(window_size_fr, len(returns_mv['ALLY'])):
    train = returns_mv['ALLY'][(end - window_size_fr):end]
    model = ARIMA(train, order=(1, 0, 1))
    model_fit = model.fit()
    pred = model_fit.forecast(steps=1).iloc[0]
    predictions.append(pred)
    actual.append(returns_mv['ALLY'].iloc[end])

actual_fr = np.array(actual_fr)
pred = np.array(pred)
valid_indices_fr = ~np.isnan(pred) & ~np.isnan(actual_fr)
actual_clean = actual_fr[valid_indices_fr]
predictions_clean = pred[valid_indices_fr]
plt.figure(figsize=(15, 7))
plt.plot(returns_mv.index[window_size_fr:window_size_fr + len(actual_clean)], actual_clean, label='Actual Returns', alpha=0.7)
plt.plot(returns_mv.index[window_size_fr:window_size_fr + len(predictions_clean)], predictions_clean, label='Predicted Returns
plt.title('Rolling Window ARIMA Forecast for ALLY')
plt.xlabel('Date')
plt.ylabel('Returns')
plt.legend()
plt.show()

mse = mean_squared_error(actual_clean, predictions_clean)
rmse = np.sqrt(mse)

print(f'MSE: {mse}')
print(f'RMSE: {rmse}')
```

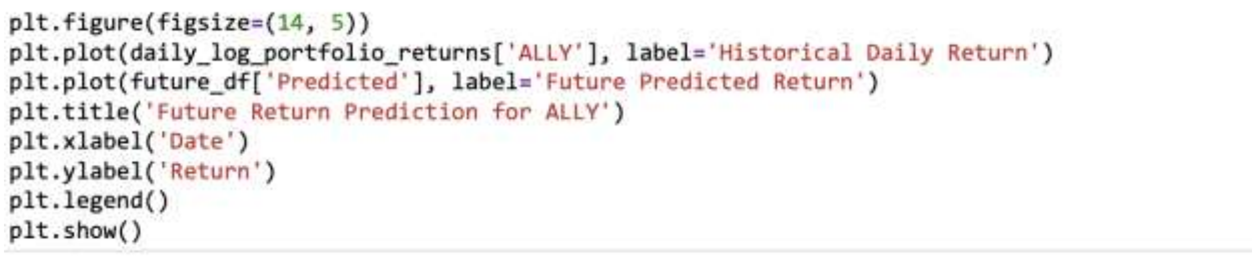
LSTM



```
data_lst = daily_log_portfolio_returns['ALLY'].values.reshape(-1, 1)
data = np.nan_to_num(data_lst)
scaler_lst = MinMaxScaler(feature_range=(0, 1))
data_scaled_lst = scaler_lst.fit_transform(data)

def create_dataset(dataset, look_back_lst=1):
    X, Y = [], []
    for i in range(len(dataset) - look_back_lst):
        a = dataset[i:(i + look_back_lst), 0]
        X.append(a)
        Y.append(dataset[i + look_back_lst, 0])
    return np.array(X), np.array(Y)
```



```
look_back_lst = 1
X, Y = create_dataset(data_scaled_lst, look_back_lst)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
X_train_lst, X_test_lst, Y_train_lst, Y_test_lst = train_test_split(X, Y, test_size=0.2, random_state=42)

model = Sequential()
model.add(Bidirectional(LSTM(100, return_sequences=True, input_shape=(look_back_lst, 1))))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Bidirectional(LSTM(100, return_sequences=False)))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

look_back_lst = 5
X, Y = create_dataset(data_scaled_lst, look_back_lst)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
X_train_lst, X_test_lst, Y_train_lst, Y_test_lst = train_test_split(X, Y, test_size=0.2, random_state=42)
model.fit(X_train_lst, Y_train_lst, epochs=100, batch_size=32, verbose=1)
future_predictions = []
current_batch = X[-look_back_lst:]
future_steps = 50

for i in range(future_steps):
    current_pred = model.predict(current_batch)
    future_predictions.append(current_pred[0, 0])
    current_batch = np.roll(current_batch, -1)
    current_batch[-1] = current_pred

future_predictions = scaler_lst.inverse_transform(np.array(future_predictions).reshape(-1, 1))
print(future_predictions)
last_date = daily_log_portfolio_returns.index[-1]
future_dates = pd.date_range(start=last_date, periods=future_steps + 1)[1:]
future_df = pd.DataFrame(future_predictions, index=future_dates, columns=['Predicted'])
```



```
plt.figure(figsize=(14, 5))
plt.plot(daily_log_portfolio_returns['ALLY'], label='Historical Daily Return')
plt.plot(future_df['Predicted'], label='Future Predicted Return')
plt.title('Future Return Prediction for ALLY')
plt.xlabel('Date')
plt.ylabel('Return')
plt.legend()
plt.show()
```