Final Report

Group Member: Zhuoran Li, Yuchen Wang, Wenxiang Gao, Zihao Chen, Congyan Yin

**Introduction**

The innovation of new energy vehicles has opened a new era for the automotive industry. The fledgling electric car industry is on the rise, and that is reflected in their stock prices as well. While investing in the stock market, the difference between the rising trend of the stock prices of electric car companies and traditional energy is a question that is worth exploring. With this question, our group selected the historical stock data of five typical auto companies. Among them, two are electric car companies, they are BYD Company Limited (BYDDY) and Tesla Inc (TSLA). The others focus on traditional energy. They are Toyota Motor Corp (TM), Bayerische Motoren Werke AG (BMWYY) and Volkswagen AG (VWAGY). We decided to finish our project with the data from the past 10 years because the more data we use to predict, the more accurate our result can be.

In order to analyze the data, we used the coding language Python. We imported them from yahoo finance to capture the most recent stock price data and pre-processed those raw data with Python’s powerful libraries such as pandas, numpy, matplotlib, etc. Then, we decided to visualize the processed data.

Visualization is a powerful tool. It can help us to gain insights into complex datasets and reveal the potential problems or the possible trends of the data. We primarily use matplotlib and seaborn to generate the charts, assisting us to understand the trends of the stock price in the past 10 years. Other than that, we employed the Autocorrelation Function and Partial Autocorrelation Function to explore the correlations between different stocks. That can help us to better understand the market dynamics.

After the pre-processing and data visualization, we used three types of different models for estimation. They are XGboost, K-nearest neighbors (KNN), linear regression, and Long Short-Term Memory (LSTM). We used those four models to predict the same brand to testify which type of model is the most effective one in estimating the data. Then we choose the best model for our prediction.

To determine how much we invest in each brand, we constructed the Monte Carlo simulation and optimized the Efficient Frontier. Through them, we would find the most ideal combination of investments to maximize our return and minimize the risk. The Capital Market Line(CML) can help us to make the best decision.

**Data pre-processing + Data visualization**

First of all, I introduced pandas and numpy modules to help me with data analysis. I then introduced two modules, matplotlib and seaborn, to help me visualize the data. In addition, I choose to introduce the yfinance module to download data directly instead of downloading data from Yahoo finance and then reading it. The advantage of this is that I can update the latest stock prices in real time. Then I introduced the pandas\_datareader module, which ensures that my downloaded data set is in the dataframe format, which will facilitate my later data processing.

When downloading the data of the five stocks, I chose the time unit as days and the time frame as ten years. When I loop through 5 stocks with the for loop, I use the globals function, which is one of Python's built-in functions that returns a dictionary of all the variables in the current global scope. This dictionary contains all defined global variables, with the key as the variable name and the value as the corresponding value. Variables in the global scope can be updated by modifying variables in this dictionary. In addition, I also used the Zip function, because its parameters are iterated objects. I use the zip function to package the data, aggregate it into the same tuple and return it, so that the tuple can keep the column with the same name.

In addition, I used a describe function to understand the basic properties of the five stock data and an info function to check if the data types are consistent. I then visualized the data with the closing prices of the five stocks, which would allow me to understand the price movements of the five stocks over a 10-year period. I found that the fluctuations were similar. In addition, I visualized the data with the daily trading volume of five stocks, which allowed me to visualize which stocks were the most actively traded over a 10-year period. Trading volume is the number of assets or securities that change hands over a period of time (usually a day). For example, stock volume refers to the number of shares traded between the opening and closing of each day. Volume, and volume over time. It is an important input for technical traders.

I also used the Autocorrelation Function and Partial Autocorrelation Function for data visualization, which helps me understand the correlation of each stock in the current month and the previous month. I find that the correlation between the current month and the current month is 1, and the longer the time interval, the lower the correlation. After that, I used the moving average method to analyze my data. moving average smoothed out price data by creating a constantly updated average price. I set three time frames, 10 days, 20 days, and 50 days. I believe that the advantages of moving averages can help traders see the trend of the stock and provide traders with more accurate trading signals.

I also used the pct\_change function to calculate the linear daily return of each stock and visualized the data. I also looked at the linear return data distribution for each stock using a histogram, and I found that five stocks had similar data distributions.

Then I used seaborn function and join plot function to compare the daily percentage return of two stocks to check the correlation between them. A correlation is a statistic that measures the relationship between two variables and must have a value between -1.0 and +1.0. Correlation measures correlation, but does not indicate causation, or causation - or whether the association is caused by a third factor. I found the highest correlation between BYD and TESLA. In addition, I used a heat map to again check the correlation between each stock's earnings and the correlation between each stock's closing prices, because I thought the heat map was the most convenient and accurate way to look at correlations.

I then calculate the mean square error of the daily return to get the volatility and expected return, which will help me check the risk of each stock. I think one of the best stocks to invest in is one with low volatility and high returns. However, I found that BMW, TOYOTA and Volkswagen were three companies with low volatility and low returns. BYD and Tesla, however, have high returns but also high volatility. After getting the volatility and expected return for each stock, I made an asset allocation. First, I calculate the daily return covariance matrix, which will be related to my later calculations. I figure out the daily return mean and then I set the weights for each stock. I also get the weighted average return through calculation, and I use the weighted average return and covariance matrix to calculate the covariance of return. Then, I set the underlying investment and a 95% confidence interval. Finally, I calculated the possible loss through ppf. The result is that there is a 95% probability that the daily loss will not exceed 26.8k USD.

**Model Selection Analysis**

**(XGboost)**

For selecting the best model, we analyze three ways to train our dataset. First, we used XGboost. The data selected is BYDDY from July 25 2013 to July 25 2023. For X value, it contains the columns 'High', 'Low', 'Open', and 'Volume' from the DataFrame df. These columns are selected as the features to be used for predicting the stock's closing price. And for y value, it contains the 'Close' column from the DataFrame df. This column will serve as the target variable for the regression task. Then we split the data to train data and test data using train\_test\_split function, making 20% of the data will be used for testing, and the remaining 80% will be used for training. Next, we train the model using the XGBRegressor function, fit the model on train data and make predictions on the test data. The prediction is stored as y\_pred. To check the model accuracy, we input r^2\_score, mse and rmse. From figure 1, we can see that the r squared score is 1, mse is 0.12, rmse is 0.37..

**(KNN)**

Next, we selected the k-nearest neighbors algorithm(KNN) as our selection testing model. The dataset is separated from the feature variables ('High', 'Low', 'Open', 'Volume') into X and the target variable ('Close') into y. Then we defined a TimeSeriesSplit object to perform time series cross-validation with two splits, using the StandardScaler to standardize the feature variables in both the training and testing sets. By performing hyperparameter tuning using cross-validation with the number of neighbors (K) as the hyperparameter, we plotted the error (mean squared error) for different values of K to help us find the best K value(see figure 2). After that, we used GridSearchCV to find the best K value from the range of values(1,150). The result for the best K value is 33. Then we created the best\_knn model using this value, and fit the best\_knn model on the scaled training data. We used the best\_knn.predict function to predict our scaled data. The graph(figure 3) plotted the actual closing prices and the predicted closing prices for the test data using the best\_knn model. We then calculated the root mean squared error (RMSE) to evaluate the performance of the model. Based on the results, our KNN model achieved an RMSE of approximately 6.35.

**(Linear Regression)**

Then, we made a linear regression model. The dataset is same for BYDDY stock. For data preprocessing, we calculated the future price using historical price data and stored it in a dataframe named “Predicted”, then dropped the last 30 rows as we don't have future data to validate these predictions. We created our X and y variables to ['High', 'Low', 'Open', 'Volume'] and ['Predicted']. Then we splitted the data to train data and test data then standardized it. After that, in order to train the model, we used the function LinearRegression() to instantiate the model, and trained the model using scaled data. For prediction, the "model.predict()" function takes the scaled test set features 'X\_test\_scaled' as input and returns the predicted target variable values (in this case, future prices) based on the trained model's learned patterns. We then plot the graph for predicted value vs actual value as figure 4A.We got the root mean squared error for 9.94, which is higher than the KNN model for 6.35, so KNN is better than Linear regression.

We still have one more model which is long short-term memory (LSTM) network. This model will be more complex, so it might be more accurate. We decided to try this model next to predict our stock price.

**Modeling And Prediction Analysis**

**（LSTM）**

This part of the project is responsible for model construction, model optimization (adjusting parameters, and forecasting stock prices for the next 30 days for 5 stocks using the best model).

For the model construction and model optimization part, in order to make the project run more efficiently, I chose the data of one stock from the five stocks to build the model and optimize the tuning parameters. For the stock price prediction part, I use the optimal model with tuned parameters to encapsulate the predictions for the five stocks.

**Data load and data preprocessing**

In order to achieve this goal, the stock of BYD Corporation (ticker symbol: BYDDY) was chosen for the study and the model was trained and tested using historical data from the last ten years. The first step of the project was to import the required Python libraries. These libraries include numpy and pandas (for data processing and analysis), yfinance (for downloading stock data from Yahoo Finance), matplotlib and seaborn (for data visualization), sklearn (for data preprocessing and model validation), and pandas\_datareader ( for reading economic data from web data sources). After setting up the required libraries, the timeframe over which the stock data was to be obtained was defined, i.e. the last ten years. Then, historical stock price data for BYD was downloaded using yfinance. Next, the "Adj Close" (adjusted closing price) was chosen as the target for model prediction and normalized. The normalization was done using sklearn's MinMaxScaler, which scales the price data to between 0 and 1. This step is to make the data of different sizes comparable in the LSTM model and also to improve the efficiency and effectiveness of the model training. In the next step of data preprocessing, the dataset is divided into training and testing sets. This is to be able to test the model with unseen data after the model has been trained, so that the generalization ability of the model can be evaluated. In the code, the choice was made to use 80% of the data as the training set and the remaining 20% as the test set. In order to convert the data into the format required for the LSTM, a function called "create\_dataset" was defined. This function accepts a dataset and a time step as input and returns a series of input data (x) and target data (y). In this function, a sliding window approach is used, where successive data of length "time\_step" are extracted from the dataset as inputs each time, and the data immediately following are used as targets. In this way, the LSTM model will be trained to predict the price of a stock on the next day after a given past "time\_step" day.

Finally, 60 and 100 were attempted as time\_steps, respectively, to generate the input and target data for the training and test sets, and the shape of the input data was reshaped into the shape required for LSTM, i.e. [number of samples, number of time\_steps, number of features].

**Model construction**

First, a basic and simple LSTM (Long Short-Term Memory) model was built and trained to predict stock prices. A Sequential model was created first, which allows for a linear stacking of neural network layers. Then, three LSTM layers were added to the Sequential model. Each LSTM layer contains 50 neurons. The outputs of the first two LSTM layers are sequences suitable for input into the following LSTM layer. The output of the third LSTM layer is a single value suitable for input into the following layer. After each LSTM layer, a Dropout layer was added with a dropout rate of 0.2 to prevent overfitting of the model. Then, a Dense layer, also known as a fully connected layer, was added, which has 1 neuron and uses a relu activation function. The Dense layer acts as the output layer of the model, outputting the predicted stock prices. After defining the structure of the model, the model was compiled. The loss function was chosen as Mean Squared Error, the optimizer was Adam, and the evaluation metric was Mean Absolute Percentage Error. Finally, the model was trained using the fit method, inputting training data and targets, setting validation data, setting 10 training epochs, batch size of 32, and printing the training process. This simple model performed quite well, with val\_mape values consistently around 13. （see Figure A)

**Model optimization**

Then the goal was to perform a parameter grid search to find the optimal parameters for the LSTM (Long Short-Term Memory) model for predicting stock prices. Multiple model trainings were carried out using the parameter grid search method, with each iteration training a new model according to the current parameter combination and then saving the model with the best performance on the validation set. This method allows for a systematic exploration of the parameter space and finds the optimal model parameter combination. First, a parameter grid was defined, including the number of neurons in the LSTM layer (50 or 100), the dropout rate of the Dropout layer (0.2 or 0.3), batch size (32 or 64), the number of LSTM layers (1 or 2), and type of activation function ('relu' or 'tanh'). The optimizer and the number of training epochs were set as fixed values ('Adam' and 10). Then, a dictionary was created to store the results. In the parameter grid, each parameter combination was iterated over. For each parameter combination, a new Sequential model was defined and LSTM and Dropout layers were added to it. The number of neurons in the LSTM layer, the type of activation function, and whether to return sequences were all set according to the current parameter combination. The dropout rate of the Dropout layer was also set according to the current parameter combination. Then, based on the LSTM layer number parameter, additional LSTM and Dropout layers were added. Finally, a Dense layer with 1 neuron and 'relu' as the activation function was added as the output layer. After defining the model structure, the model was compiled. The optimizer was set as 'Adam', the loss function was set as 'mean\_squared\_error', and the evaluation metric was set as Mean Absolute Percentage Error. Next, a ModelCheckpoint was defined, which is a callback that saves the model at the end of each training epoch. The ModelCheckpoint saves the model with the best performance based on the Mean Absolute Percentage Error of the validation set. Finally, the model was trained using the fit method. Training data and targets were inputted, validation data was set, the number of training epochs was set, the batch size was set, and the ModelCheckpoint callback was added. (see Figure B)

**Model evaluation**

Next the time series data was predicted using the trained machine learning model and the prediction performance was evaluated by calculating the Root Mean Square Error (RMSE) metric. First, the required matplotlib library was imported for data visualization. Then, the input data from the training and test sets were predicted using best\_model. To reduce the predictions to the original data scale, the inverse\_transform method of the scaler object was used. Next, the RMSE between the prediction results of the training and test sets and the true target value is computed by importing the math library and the mean\_squared\_error function of the sklearn.metrics module.Finally, the RMSE values are exported in order to evaluate the prediction accuracy of the model and to infer the model performance from it. And show it with data visualization

**Single stock prediction**

The goal of this code is to use a pre-trained model to forecast stock prices for the next 30 days. Firstly, two variables, time\_step and future\_days, are defined to set the time step and the number of days to predict in the model. Next, the 'Close' column is extracted from the original data set df as the target for prediction. Then, a new dataframe new\_df is created, containing only the last time\_step days of stock prices as input for the model. The data is normalized using a MinMaxScaler object, scaling it to a range between 0 and 1, which improves convergence and performance for neural network models. Subsequently, a loop is used to make predictions for the next 30 days. In each iteration, the last time\_step days of normalized stock prices are taken and reshaped into a three-dimensional input tensor (samples, time steps, features) as required by the neural network model. The pre-trained model is then used to predict the next day's normalized price. The predicted price is then transformed back to its original scale using inverse normalization, resulting in the forecasted stock price for the next day. After each iteration, the new predicted price is added to scaled\_data, updating it for the next prediction. To maintain the fixed time step, the oldest data is removed. Once the loop is completed, all 30 days of predicted stock prices are stored in the predicted\_prices\_df dataframe along with the corresponding dates. The date range starts from the last date in the original data set and extends continuously for 30 days. In conclusion, predicted\_prices\_df contains the forecasted stock prices and their dates, which can be used for analysis and making investment decisions for the next 30 days. (see Figure C)

**Packaged Data Preprocessing**

The aim is to define a process\_data function to mainly preprocess the stock data to adapt it to the subsequent training and prediction of the LSTM model. First, the function selects 'Close' (closing price) as a feature and normalizes it using a MinMaxScaler so that all the data is in the range 0-1. This is because neural networks are better suited to handle data in this range. The function then divided the data into a training set and a test set, with 80% of the training set and 20% of the test set. This is to ensure that the model has both enough data for learning and data for testing the model's effectiveness. Next, the function calls the internally defined create\_dataset function to convert the data from the training and test sets into the format needed for the LSTM model. This function divides the continuous stock price data into a series of samples, each of which contains continuous time-step data, as well as the target value corresponding to that sample (the data for the next time step). Finally, the function adjusts the sample data to the three-dimensional format required by the LSTM model. This is because the LSTM model requires the input data to be in the format [samples, time\_steps, features], which represent the number of samples, the time step for each sample, and the number of features for each time step, respectively. The function returns the processed training set, test set, and a scaler object for normalization. the scaler object is used for subsequent anti-normalization operations to convert the predicted data back to the original stock price ranges.

**Packaged Prediction**

The objective is to define a `predict\_future\_stock\_price` function to be used to predict stock prices for the next few days. First, the function creates a new dataset containing only 'Adj Close' data (adjusted closing price) for the past `time\_step` days. This data will be used as input to the model. Next, the new dataset is normalized using the `scaler` object of the previous training data so that the data is between 0 and 1. This is to keep the data consistent and allow the model to process the new data correctly. The function then performs a loop to predict the stock price for the next `future\_days` days. In each loop, data from the past `time\_step` days is used to predict the price for the next day. The predictions are back-normalized, converted back to actual stock prices, and added to the list of predicted results. Finally, the predicted price is added to `scaled\_data` for the next prediction. The function finally returns the list of prediction results, i.e. stock prices for the next `future\_days` days. These results can be used directly for analysis or visualization to understand the future stock price trends predicted by the model.

**Download the data And Predict 5 stocks**

A list of stock symbols to be predicted, stocks\_symbols, is first defined, and then two dictionaries are created for the stock data and the prediction results. Then, for each stock symbol in the list, the code downloads the corresponding stock data and preprocessing the data (normalization, training-test set partitioning, etc.). The processed data is fed into the model for future stock price prediction. The prediction results are stored in the stocks\_predictions dictionary. Afterwards, the code converts the predictions dictionary into a dataframe predicted\_prices\_df for subsequent analysis and visualization. Then, the code creates a date range to represent the predicted future dates and adds this date range to the predicted results dataframe. Finally, the code sets the index in the predicted\_results dataframe to a date to facilitate subsequent viewing of the predicted results by date. (see Figure D)

**Portfolio Allocation**

This section conducts a portfolio optimization analysis based on historical stock data for these five major automakers: BYD Company Limited (BYDDY), Toyota Motor Corp (TM), Bayerische Motoren Werke AG (BMWYY), Tesla Inc (TSLA), and Volkswagen AG (VWAGY). The data was fetched from Yahoo Finance for a ten-year period from July 25, 2013, to July 25, 2023.

This study chooses to use ten years of data, since the purpose of this study is to provide a tool that everyone can use, instead of merely providing a setup for the investment portfolio position for the next month. If the demands are making corresponding position settings based on the price predicted in the next month or more. It is only necessary to replace the data of " the ten-year stock data of these five companies" in the code with forecast data.

**Individual Asset Analysis:**

Before constructing the investment portfolio, this study first selected logarithmic returns for each stock's daily performance. (*See Figure 5*)Compared to simple returns, which represent absolute returns for each period, logarithmic returns represent relative returns. Using logarithmic returns for calculating the portfolio's performance eliminates the impact of changes in asset size on the return calculations. Next, the study employed the Sharpe ratio and visualized it for a simple comparative view, laying the groundwork for portfolio optimization. The Sharpe ratio provides a method to compare investment returns with their risk. The higher the Sharpe ratio, the better the historical risk-adjusted performance of the asset.

In this study, all code was written as functions as much as possible (including but not limited to annualized returns, standard deviation, etc.) to facilitate future code optimization.

**Portfolio Analysis:**

To search for the investment portfolio, the first step in this study involved constructing a simple Monte Carlo simulation. In the model building process, random weights were selected for each asset, with the constraint that the sum of all weights equals 1. The purpose of this constraint is to prevent negative weights or "short-selling," as it may not always be feasible for investors to borrow an asset, sell it immediately, and then buy it back at a lower price in the future for return.

After defining functions to calculate the portfolio's expected return and volatility using matrix methods, these functions were called and utilized with the generated weights to perform the calculations. The Monte Carlo simulation was conducted by repeating this process multiple times (n times). (*See Figure 6*)The visualization clearly depicts each point on the chart as a representation of one particular portfolio. (*See Figure 7*)

Now, the question arises - which portfolio is more optimal?

**Efficient Frontier:**

The second step in this research is to define the Efficient Frontier.(*See Figure 8*) The Efficient Frontier represents the trade-off between return and risk for different combinations of investment assets. Each portfolio on the Efficient Frontier is either superior or at least as good as other portfolios, forming a boundary. Portfolios on the Efficient Frontier are considered efficient because they offer the highest expected return for a given level of risk or the lowest risk for a given level of expected return. This optimal set of risk-return trade-offs is obtained through the process of optimizing different asset allocation schemes. (*See Figure 9*)

**Global Minimum Variance Portfolio (GMVP):**

To further optimize the portfolio, this research chooses to identify the Global Minimum Variance Portfolio. The characteristic of the Global Minimum Variance Portfolio is that it has the lowest risk level among all possible investment portfolios. This means it exhibits the minimum volatility compared to other portfolios. The Global Minimum Variance Portfolio is located at the far left of the Efficient Frontier, representing the lowest risk level. While the Global Minimum Variance Portfolio offers an ideal investment option in terms of risk reduction, it may not necessarily be the best choice for investors as its expected return may be relatively lower. (*See Figure 10*)

**Capital Market Line (CML):**

However, as mentioned earlier, not every investor will necessarily choose the portfolio with the lowest risk. Therefore, it is essential to plot the Capital Market Line (CML) to represent portfolios composed of risk assets and risk-free assets. When calculating the CML, the risk-free rate is chosen as the U.S. three-month Treasury bill rate of 0.054%. (*See Figure 11*)

Based on the data of ten years stock of these five companies, by finding the optimal investment allocation portfolio, the approximate asset allocation weights for BYD, Toyota, BMW, Tesla and Volkswagen are respectively about 3.25\*e^-17, 2.73\*e^-2, 9.73\*e^-1 and 3.63\*e^-18.

The CML is a straight line in the capital market, representing portfolios formed by combining risk assets and risk-free assets at different proportions. Each point on the CML represents a specific portfolio with varying allocations between risk assets and risk-free assets. All portfolios on the CML are located on the Efficient Frontier, meaning they represent the highest expected return for a given level of risk.

The key characteristic of the CML is that, for a given risk-free rate, it offers the highest Sharpe ratio among all efficient portfolios. The Sharpe ratio measures the excess return obtained by the portfolio for each unit of risk it assumes. Therefore, portfolios on the CML are considered optimal because they achieve the maximum excess return for a given level of risk.

The slope of the CML represents the trade-off between risk assets and risk-free assets. When the slope is steeper, investors can obtain higher expected returns by increasing the proportion of risk assets in their portfolio, but they also take on higher risk. Conversely, when the slope is gentler, investors choose to increase the proportion of risk-free assets to reduce overall risk, but the corresponding expected return will be lower. It can be seen from the figure that due to the current high interest rate of US treasury bonds, the slope of CML is still relatively flat. (*See Figure 12*)

**Conclusion**

According to the research above, we found that the electric vehicle companies are in a significant increase. In the pre-processing and visualization part, we found that the linear return of those five stocks has similar data distribution, but Tesla and BYD have the higher ranger in the linear return, meaning that they have higher “potential”. Also, in terms of volatility and expected return, BMW, TOYOTA, and Volkswagen have low volatility and low return, other stocks are both having high returns but high volatility. It has represented that the new energy vehicle companies possibly have higher investment return but also comes with a higher risk of losing money compared to BMW, TOYOTA, and Volkswagen.

Then, we used four models to testify our idea. In those models, LSTM and XGboost models have the highest performance in accuracy. XGboost has a high R-squared value, low mean squared error, and low root mean squared error, having high accuracy on the results. The KNN model has the worse RMSE value compared to XGboost, and the linear regression has the worst return. Its RMSE value is 9.94. However, along with our testing, we found that the LSTM model is the most effective.

For the LSTM model, we used the data set of BYD to train the model to make the model fit the data better. Then, we used the model trained by BYD to apply to other companies to get the estimated stock price in future 30 days.

For the portfolio allocation part, our Monte Carlo simulation and Efficient Frontier constructed different combinations of asset weights. Then we used Capital Market Line to guide us determine the optimal combination of risk assets and non-risked assets. Therefore we can achieve the highest Sharpe ratio.

According to the decision we made in portfolio allocation, we decided to invest 97.3% of our money into Tesla and 2.7% of our money into Toyota because that is the investment combination we found to be the most profitable with the least risk.

Figure 1

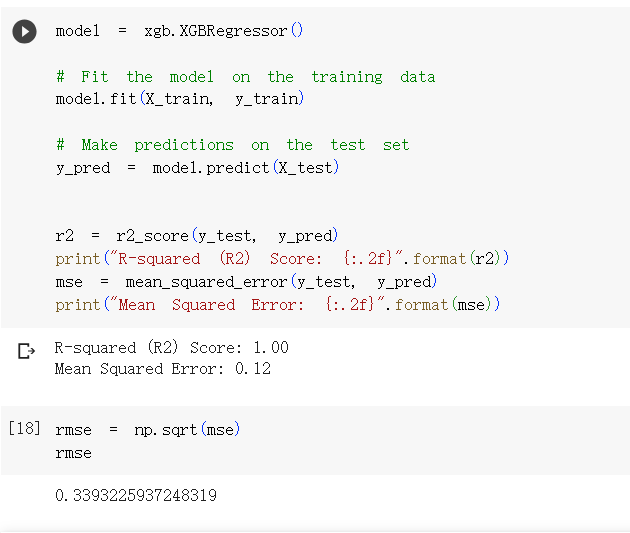
****

Figure 2

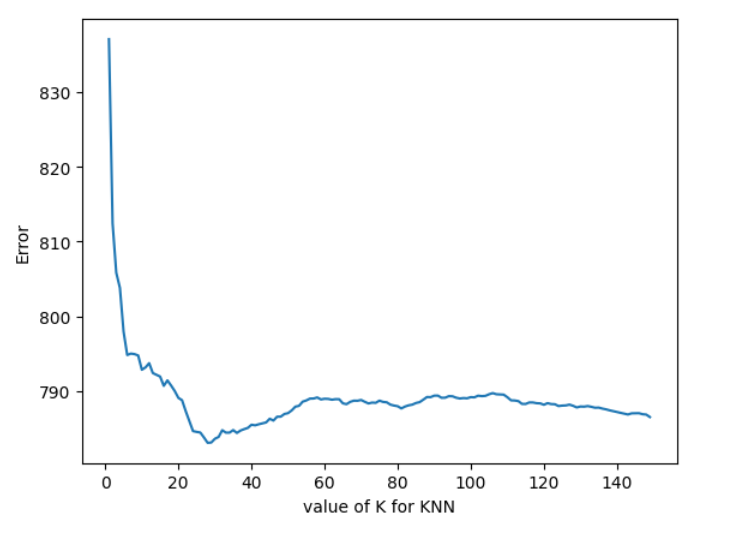


Figure 3

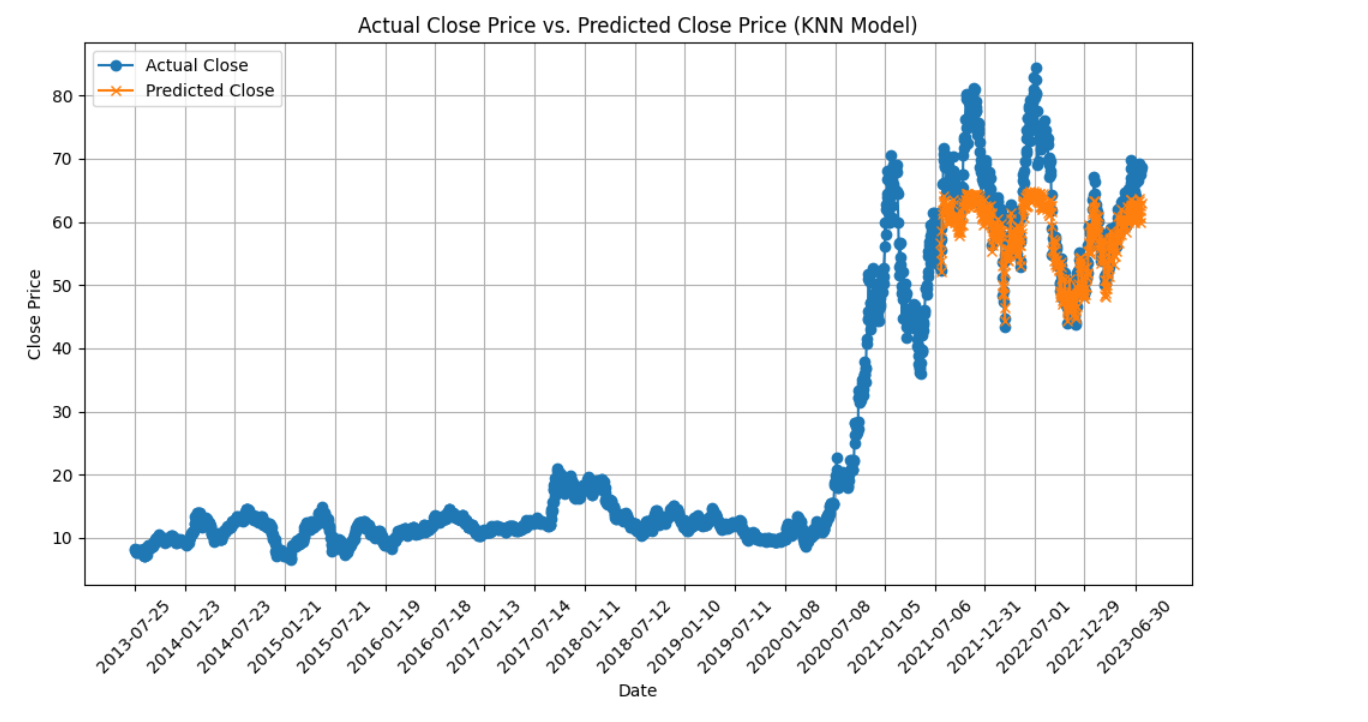


Figure 4

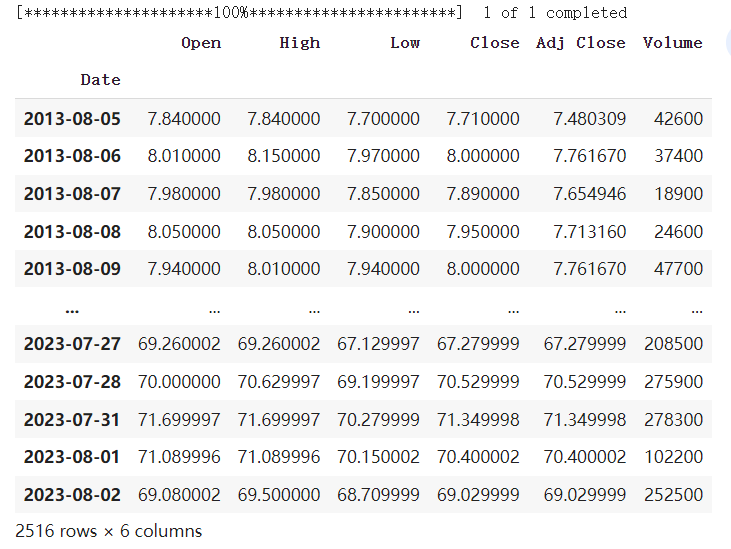
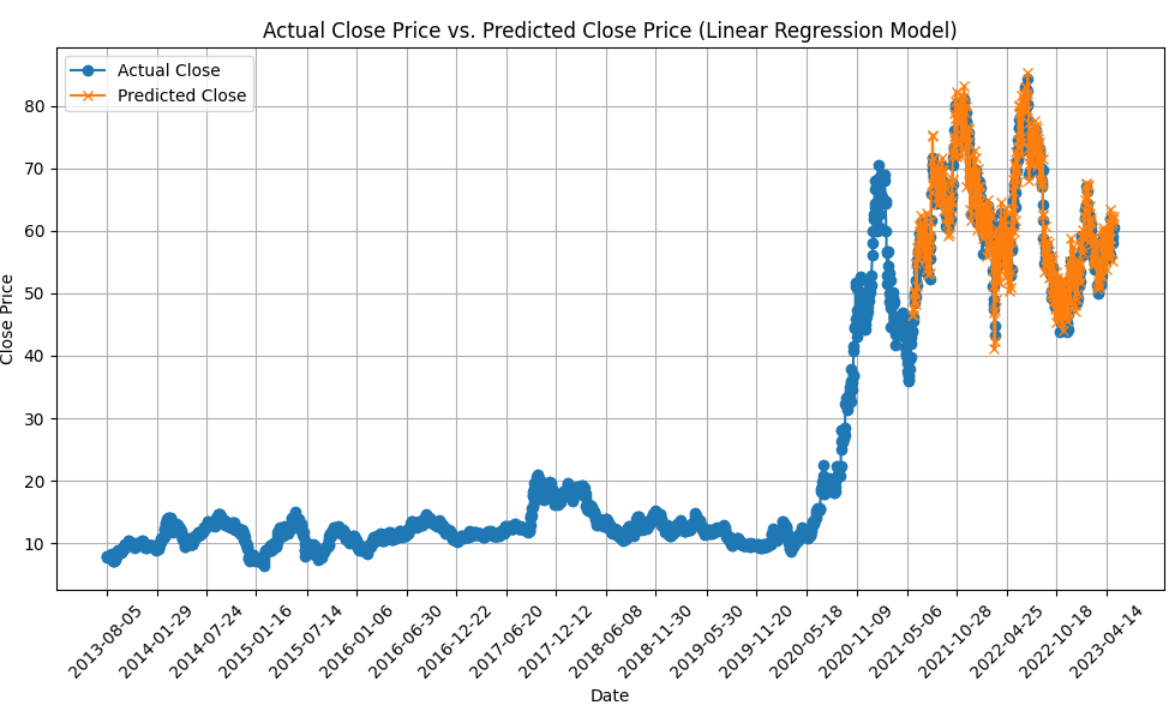
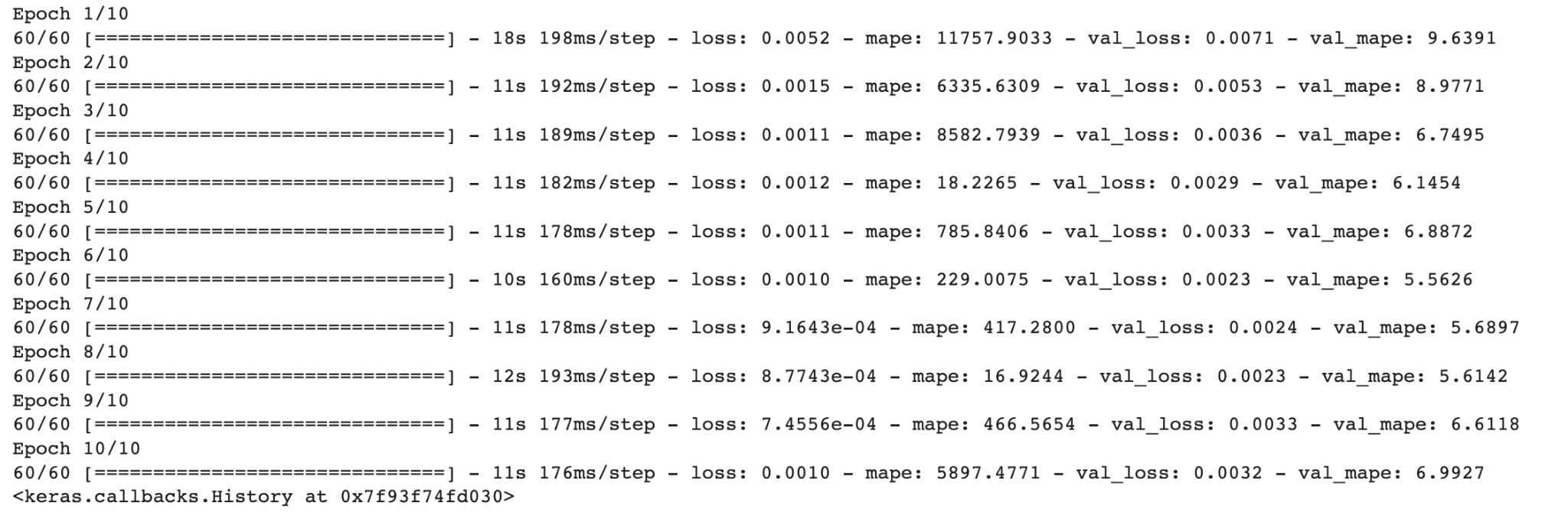


Figure 4A





FigureA

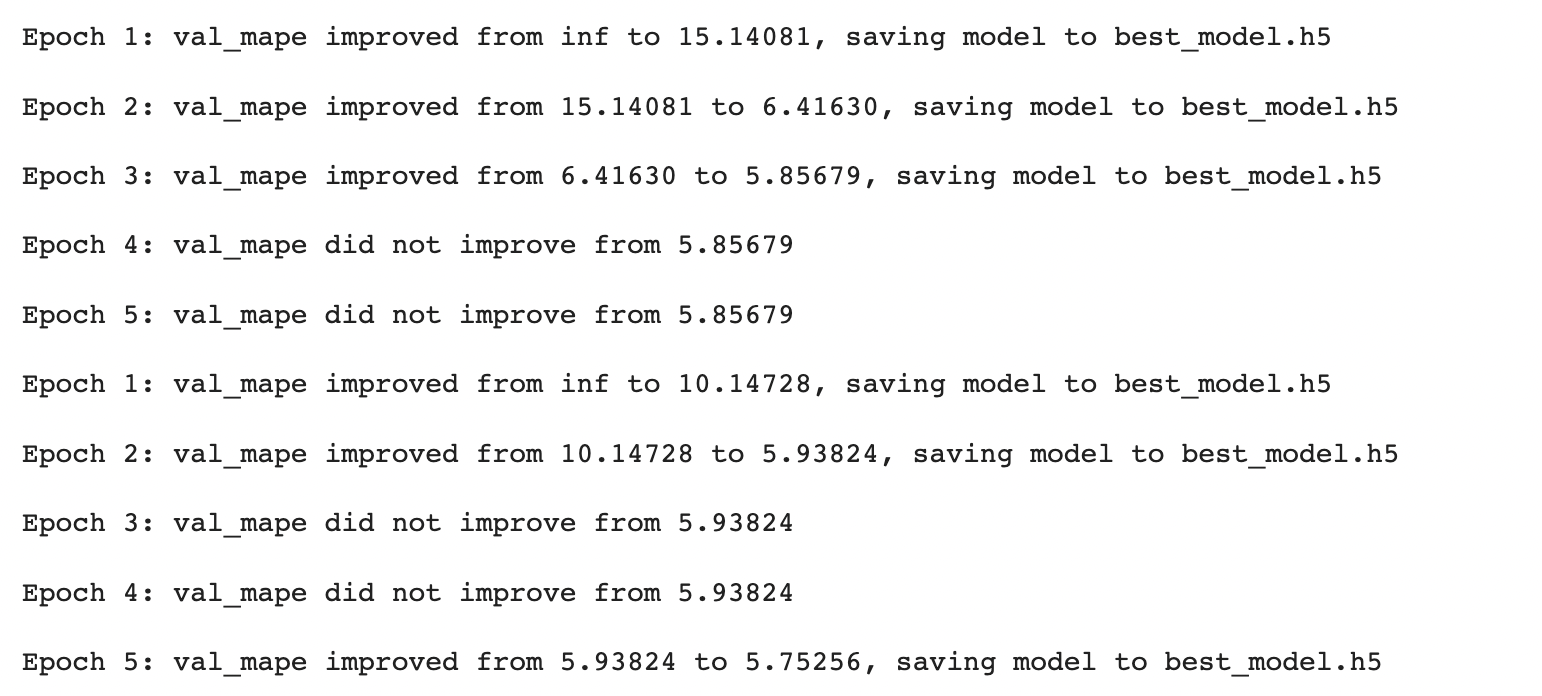


Figure B

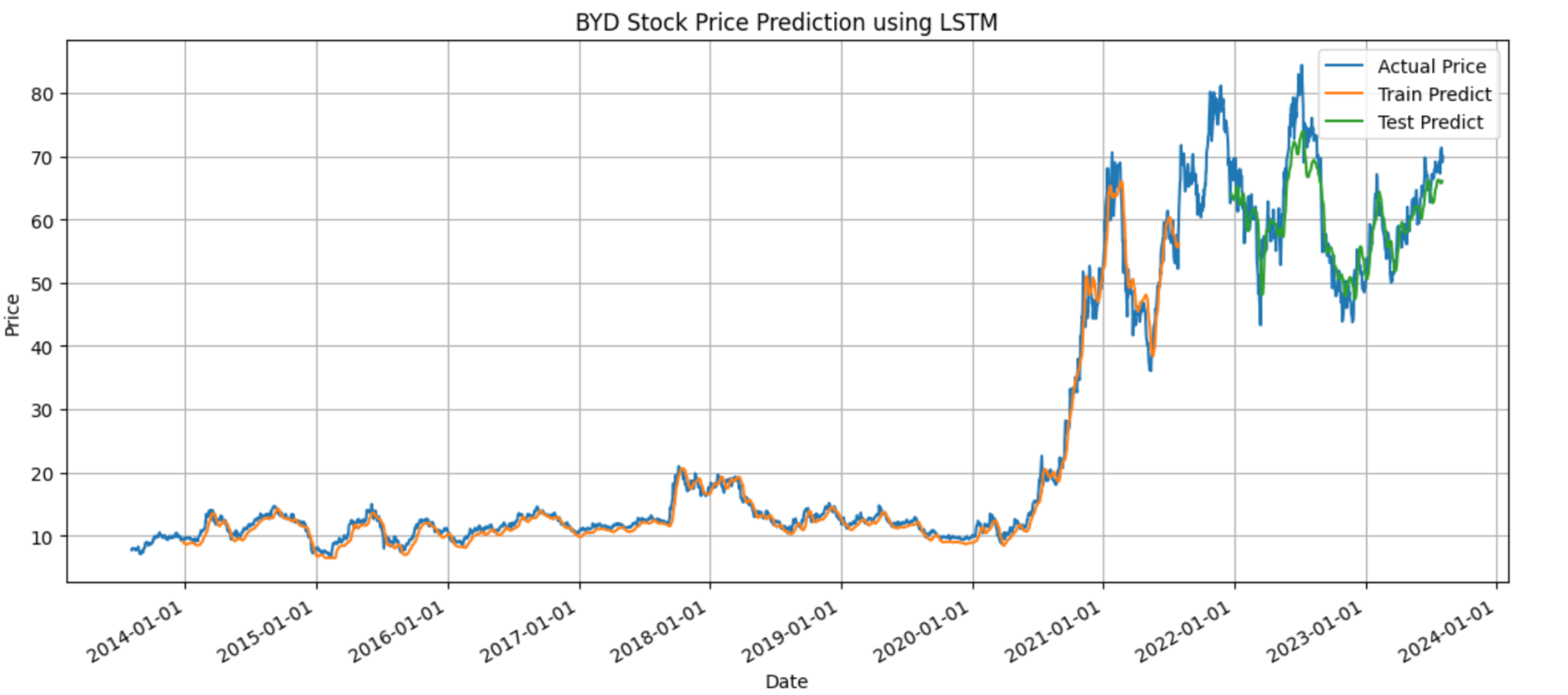


Figure C

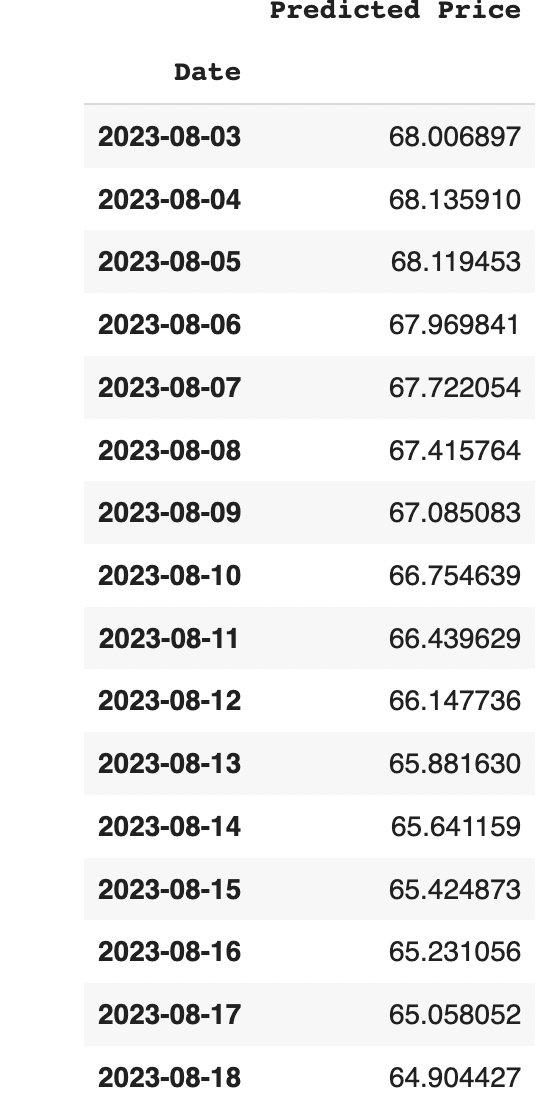


Figure D

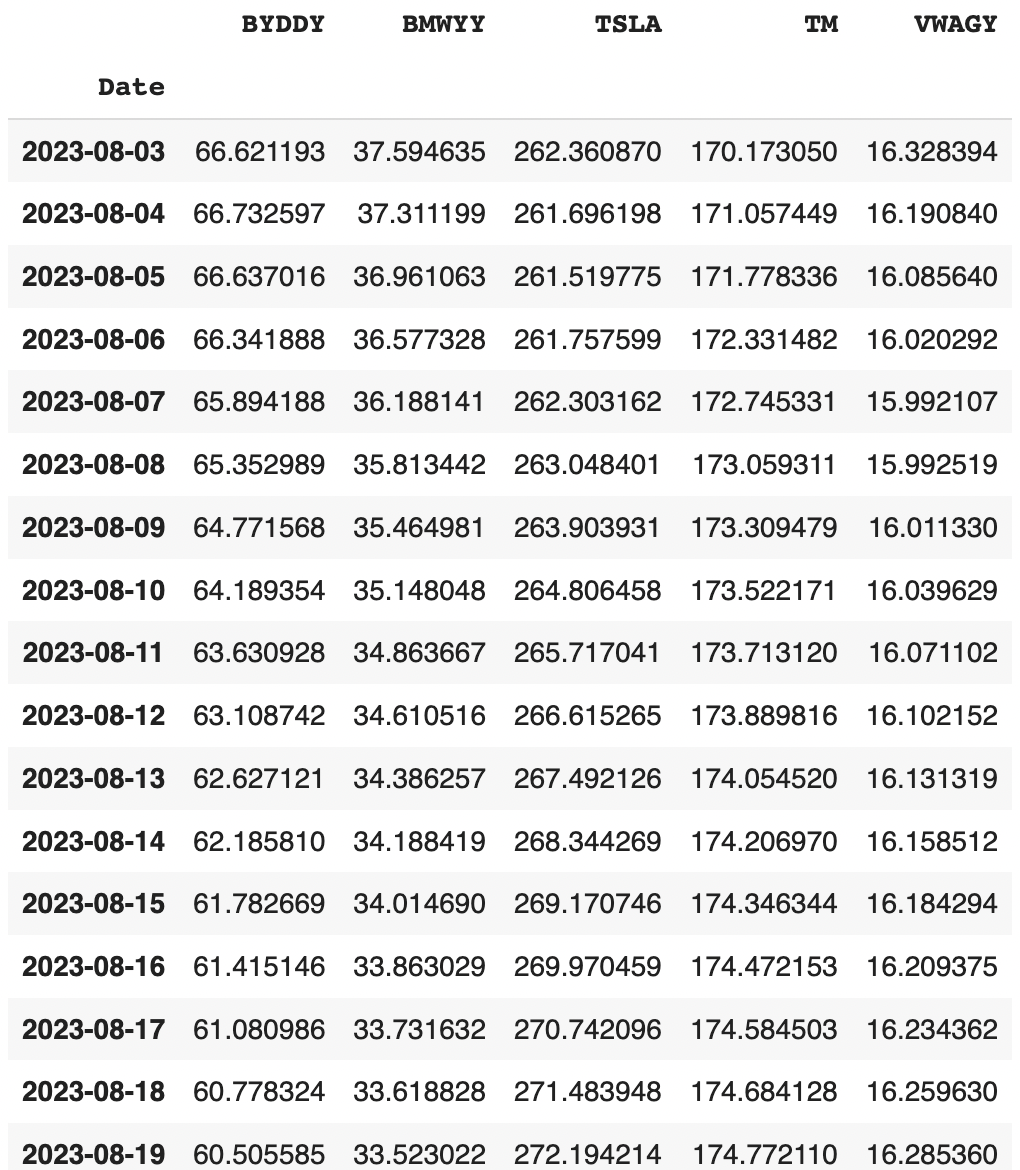


Figure E

Figure 5

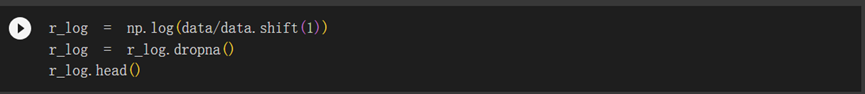


Figure 6

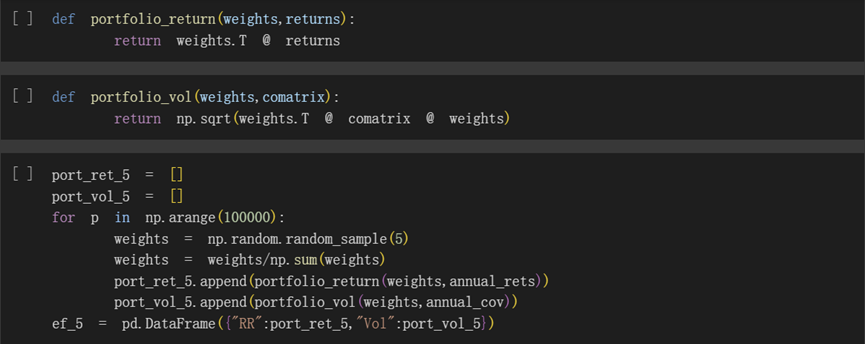


Figure 7

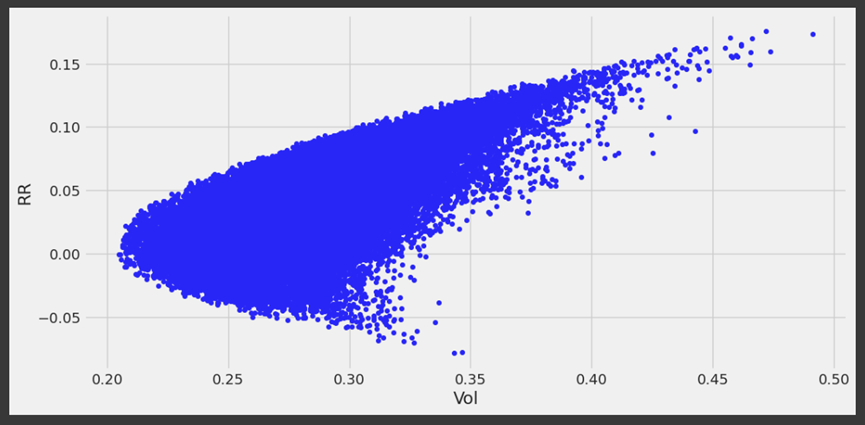


Figure 8

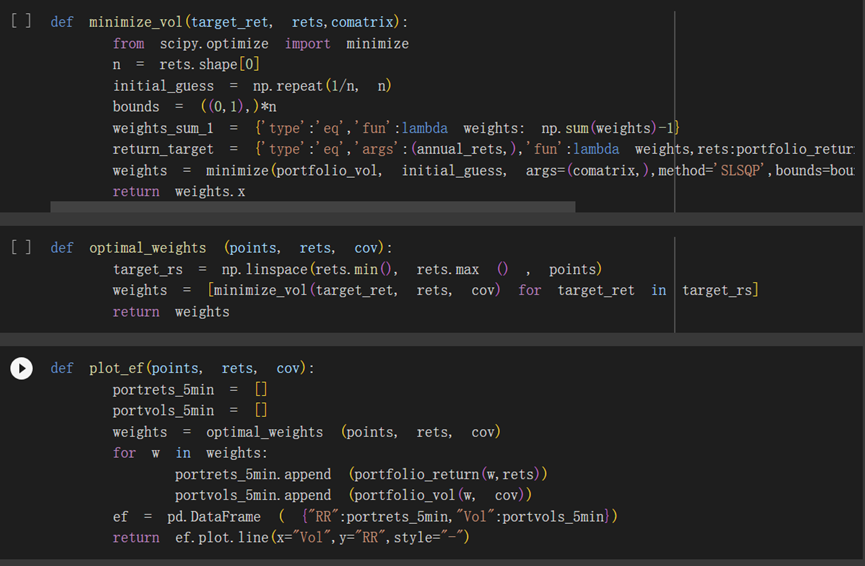


Figure 9

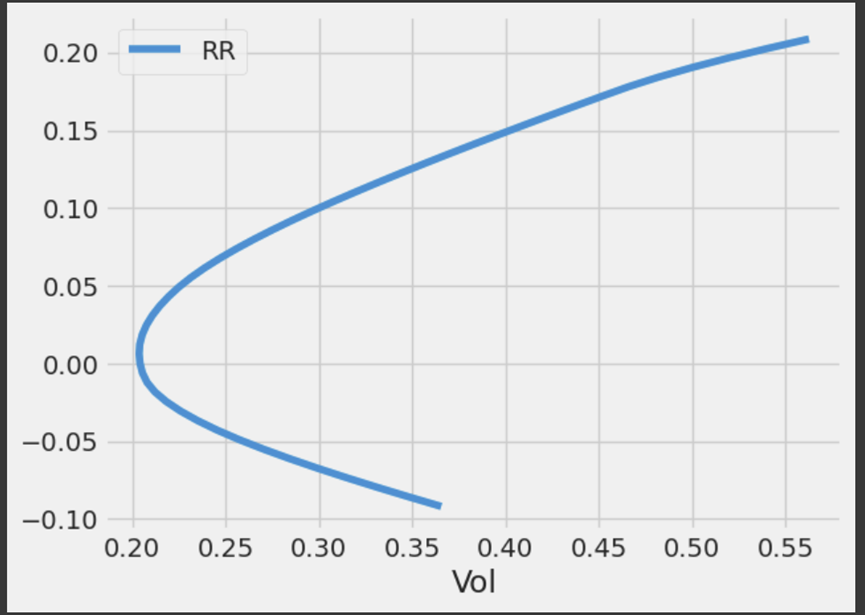


Figure 10

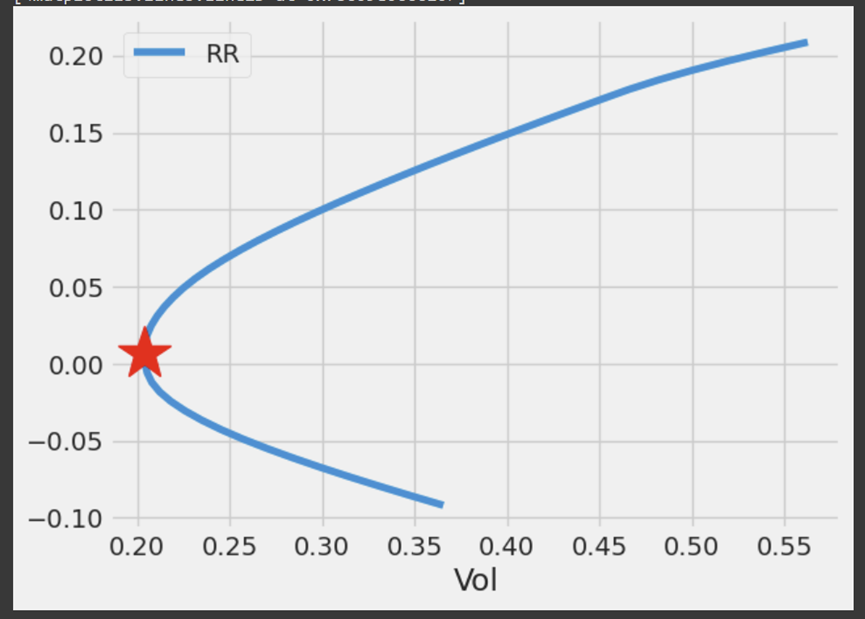


Figure 11

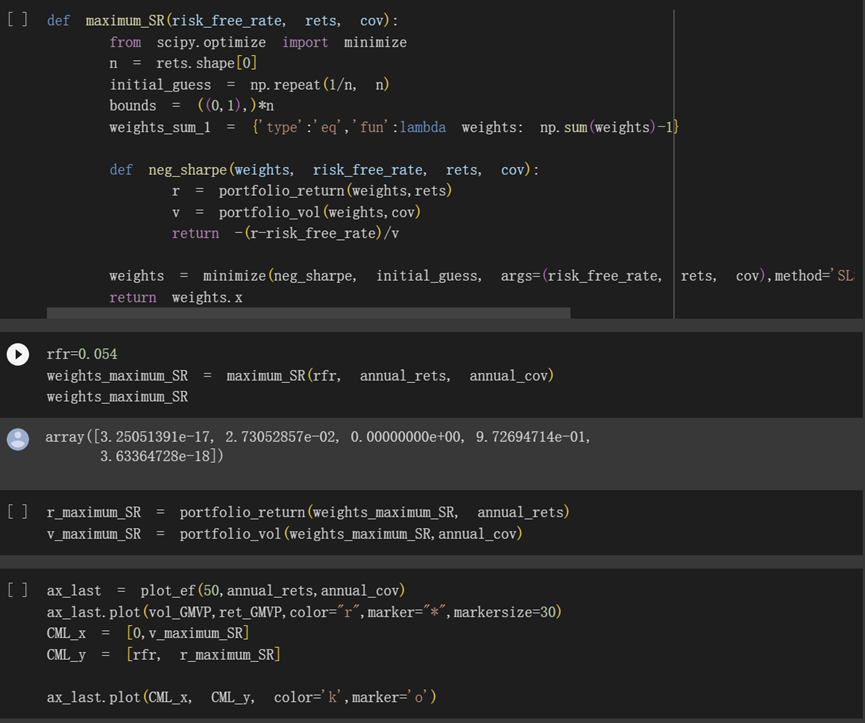


Figure 12

