Stock Price Prediction Using Machine Learning

John Langenderfer, Addison Morgan, Dawin Ye, Andrew Shen

1 Background

This project aimed to develop a machine learning model capable of accurately predicting stock prices over time, a task that has always been challenging due to the complex and dynamic nature of financial markets. The stock market is influenced by numerous factors such as global events, economic indicators, investor sentiment, market shocks, and idiosyncratic risks, which are unique to individual companies or industries.

There are two primary approaches to stock predictions: fundamental and technical analysis. Fundamental analysts focus on qualitative aspects, while technical analysts rely on firm, price, and volume data. The Efficient Market Hypothesis, a widely discussed theory, suggests that stock prices accurately reflect all available information, making it very difficult to try to beat the market. However, quantitative hedge funds like TwoSigma and Renaissance consistently outperform the market, indicating that a competitive advantage may be attainable.

To address the complexities of stock prediction, our team leveraged advanced techniques in data preprocessing, feature engineering, and model selection. We aimed to build a predictive model to assist investors in making informed decisions and improving their investing performance. To mitigate the impact of idiosyncratic risk, investors often diversify their portfolios using a "basket approach," spreading risk across multiple assets.

Each team member chose a different model to work on, discussing and comparing the results. Dawin focused on linear regression, Andrew on random forest, AJ on SVR, and John on LSTM. Accurate stock price predictions can provide valuable insights for investors and traders, help them make informed decisions, construct effective trading strategies, and potentially automate the investment process.

In this work, we aim to develop a model to predict the closing day stock prices for the next day and attempt to make an efficient investing strategy.

2 Approach

Technical analysts operate on the fundamental assumption that market patterns exist and that upward and downward trends can be identified. Ideally, the most effective models should capitalize on these large-scale trends and accurately predict when price shifts will occur. Our team gathered price and volume from Yahoo Finance, dating back to various time periods, as the model's performance varied based on the timeframe. The data included each day's Open, High, Low, and Close prices, as well as the volume traded.

To ensure that the model could produce accurate results and use those outcomes as input for subsequent day's predictions, we opted to focus on the Closing Price as the primary input. We

experimented with varying the number of days of closing prices fed into the models. This approach enabled us to use the predicted closing price as the most recent day's closing price, allowing the model to make predictions for the following day, and so on.

By refining our methodology and emphasizing the Closing Price, we aimed to enhance the model's overall effectiveness and reliability, ultimately providing valuable insights for investors and traders to make informed decisions in the ever-changing stock market.

3 LSTM

The LSTM (Long Short-Term Memory) model is a type of recurrent neural network (RNN) that can handle long-term dependencies by selectively retaining or forgetting information. The model achieves this by using three gates - input, output, and forget - that are controlled by sigmoid functions to determine how much information to keep or discard at each time step. The loss function used is mean squared error, and weights are adjusted using backpropagation through time.

The initial LSTM model used for this project is a single-layered stacked LSTM model with 128 LSTM units in the first layer and 64 units in the second layer. The model was compiled using the Adam optimizer and mean squared error as the loss function.

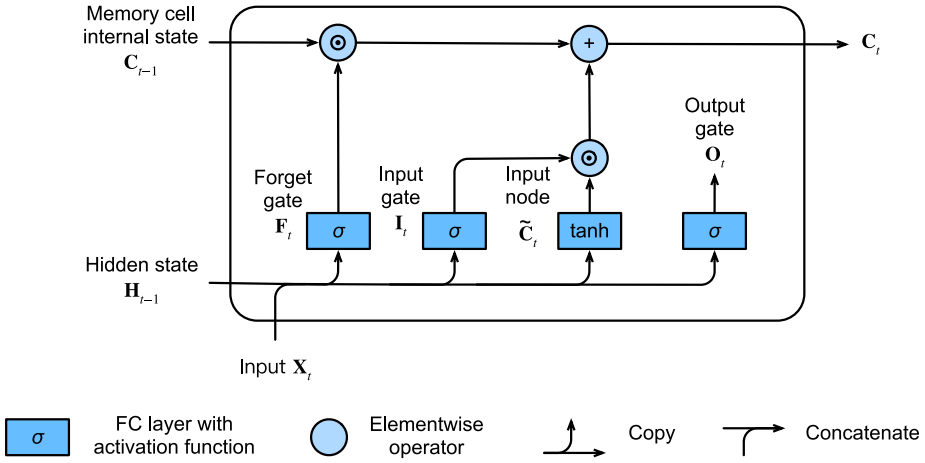

Figure 1: Single LSTM unit

The LSTM equations are given by:

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\
\tilde{C}_t &= \tanh(W_C[h_{t-1}, x_t] + b_C) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \tanh(C_t)
\end{aligned}
$$

The sigmoid function is used in the forget, input, and output gates because its output ranges from 0 to 1, which allows the gates to control the flow of information in a proportional manner. When the sigmoid function's output is close to 0, it blocks information flow, and when it's close to 1, it allows information to flow.

The tanh function is used for the candidate cell state and the final hidden state update because its output ranges from -1 to 1, providing a normalized representation of the internal values.

We can calculate the loss using the Mean Squared Error (MSE) between the predicted value ($\hat{y} = h_t$) and the actual value (y): In a typical sequence prediction problem using an LSTM, the input data is a sequence of length T, and the model generates a sequence of N predictions.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

In our stock LSTM algorithm, we a predicting the closing price for the next day based off the previous 60 days We then calculate the loss using the output from that, and the actual closing price In our case, we are predicting a single number, so the equation would be

$$\text{MSE} = (y_i - \hat{y}_i)^2$$

To update the weights and biases, we compute the gradients and adjust the weights using stochastic gradient descent. In practice, the gradient calculations involve more complex partial derivatives, and we would use a deep learning framework to compute them automatically using backpropagation through time.

After updating the weights and biases, the LSTM would go through the sequence again, and the process would be repeated multiple times (epochs) until the loss converges to a minimum value.

3.1 Data Collection

The stock data was collected using the Yahoo Finance API in Python. The dataset includes the daily open, close, high, low, and volume data for the selected stocks. The dataset also includes additional features such as moving averages, MACD, and RSI. We did feature selection, although found just the closing price to work the best. For the LSTM model, it was trained on data for each of 30 different stocks since 2014, and then tested on the last year.

3.2 Training and Validation

The data was split into training and a validation set. The training set consisted of 89 percent of the data (2014-2022), while the validation set and test set each contained 11 percent (the last year). The model was trained on the training set, and the performance was evaluated using the validation set. Predictions were made using a sliding window of 60 days (SVR will be 20 days), meaning that each prediction was based on the previous 60 days of data. This approach was chosen to capture temporal dependencies in the data and improve the accuracy of the predictions. The data was only trained for 1 epoch, as the loss quickly converged. (Figure 2)

3.3 Data Preprocessing

The data was preprocessed to remove any missing values and to normalize the data. Normalization was performed to ensure that each feature was on the same scale and to prevent any one feature

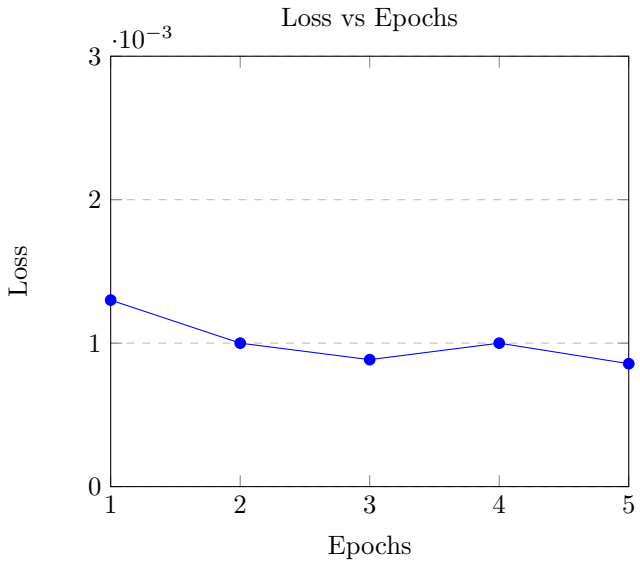

Figure 2: Loss values for each epoch

from dominating the others. The dataset was then split into training and testing sets. The training set consisted of the first 80 percent of the data, while the testing set consisted of the remaining 20 percent.

3.4 Model Architecture

Two types of LSTM models were implemented, namely, the normal LSTM and the bidirectional LSTM. Both models consisted of two hidden layers with 128 neurons each, followed by a dense output layer with a single neuron. The normal LSTM model takes in a sequence of data points and processes them sequentially, while the bidirectional LSTM processes the sequence in both directions.

Regularization techniques were used to prevent overfitting of the model. Dropout and L2 regularization were applied to both the normal and bidirectional LSTM models. Dropout randomly drops out a certain percentage of neurons during training, while L2 regularization adds a penalty term to the loss function to encourage smaller weights.

Additional features were added to the model to improve its accuracy. These features included the MACD (Moving Average Convergence Divergence) and RSI (Relative Strength Index). The RSI measures the strength of a stock's price action, while the MACD measures the difference between two exponential moving averages. These features were calculated using the `ta` (Technical Analysis) library in Python. The models were tested with the addition of these features, both with the initial LSTM architecture and with the bidirectional LSTM architecture with regularization. Some additional features were considered, although not implemented such as sentiment analysis from news sources, due to EMH.

Hyperparameter tuning was performed to optimize the performance of the models. Changing the number of epochs was found to lead to overfitting, as the loss function quickly converged. Dropout and L2 regularization were added to prevent overfitting. The number of hidden layers and

neurons per layer was also experimented with and did not have a positve effect.

3.5 Trading Algorithm

Our evaluation of the models was difficult to do. MSE did not provide much insight, except for which model to choose for the trading algorithm. Based off the nature of the LSTM, and as you will see with other models, it is better at predicting overall trends, so using MSE or seeing if they both moved in the same direction the next day turned out with poor results. Thus, we came up with this trading algorithm based off the next day predictions, and only implemented it with LSTM as it had the best trend predictions.

To test the effectiveness of the LSTM model, we created a trading algorithm that buys and sells stocks based on the predicted prices for the next day. The algorithm was tested on a list of 30 stocks from different industries over the past year, with the aim of producing a profitable trading algorithm.

The trading algorithm follows these rules:

- If the predicted price for the next day is greater than the current price, buy the stock.
- If the predicted price for the next day is less than the current price, sell the stock.
- If you sell and the next day's price is less, hold.
- If you buy and the next day's price is more, hold.

The output of the performance is given by the total profit or loss across the 30 stocks, assuming an equal distribution of 100 dollars per stock in the portfolio. Below is an example for TSLA, although this was done for 30 stocks over the last year and results shown below in the results section.

Figure 3: Example of trading algorithm for TSLA

4 SVR

For the SVR model, a different approach was needed than for the LSTM model, as Support Vector Machines can do better with less data given. Instead of its input data being 60 days, it worked better down to 20 days or less. It was also only using about two years or less of data in training and testing, unlike the years of data in the LSTM approach. This was to deal with overfitting, as using larger amounts of data caused it to heavily rely on the previous day's closing price for the prediction. On paper, that does not sound too bad, but the main problem was the output graph for the test data was almost the exact same as the real data, just shifted over a day, meaning the model was always a day late on the prices. Another way to deal with overfitting was to use Grid Search and find a good value for the C regularization parameter, as well as doing a grid search on the kernel and gamma parameters. These tended to be a linear kernel and the "scale" gamma parameter, which uses a gamma of 1 / (num-features * Var(X)). This parameter was only useful if a different kernel was used, though, so it can be disregarded, but it is kept in the grid search in case a change in the gamma helped a different kernel work better than the linear one for some subset of the data. The C value changed the most, depending on the length and starting date, while the kernel and gamma stayed the same for the most part, as the data did not need to be transformed much to work with the SVR. As compared to a linear regression model given the same amount of data, the SVR model outperformed it up to about this length of data given, which was expected due to Support Vector Machines being a more useful model given less data.

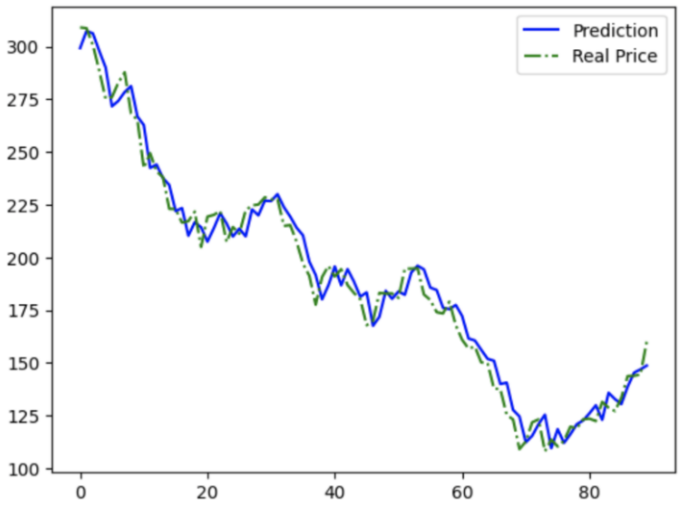

Figure 4: Prediction vs. Real Price

It was able to follow along with the test data for the most part accurately, and when it came time to test the model feeding back into itself, it stayed close to the true data.

The model over these predicted days does follow along with the true prices, and loses its accuracy towards the end. This is due to the weight the model puts on the "day before" price that is its own prediction, so the further the prediction is from the last true date, the less the model is able to accurately predict, as it feeds its results back into itself for the next day's prediction.. With this same data, the Linear Regression model had about the same test score, but its future predictions

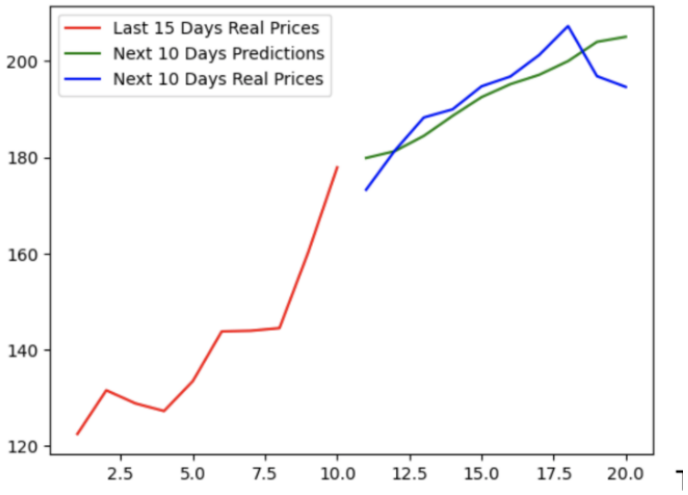

Figure 5: Prediction vs. Real Price

strayed further away from the true future prices. A different SVR model called NuSVR on SKLearn was implemented, with the nu parameter controlling the number of support vectors, but the results for it were about the same or worse than the normal SVR model, so it was removed. As for the model itself, this would only be a superior model for a relatively new stock in a new market, as other models work better when they have more data, including access to other stocks in the same market and just more years worth of data.

5 Linear Regression

In the Linear Regression model, we used SKLearn's library to find a linear relationship between a company's stock price tomorrow and the stock price of the previous N days. To determine the optimal number to use for N, a brute force approach was used to find which N between 5 and 99 minimized the MSE for that particular stock. The reason numbers lower than 5 were not considered were because the predictions often looked like a straight line, which meant it was underfitting the data as it couldn't model the complexities of the stock market. Overfitting appeared to occur even earlier than 99, so numbers beyond that did not need to be considered either. After using linear regression, we used Ridge regression to improve the model by adding an alpha term as a regulator, which was found by using GridSearchCV, a library built for hyperparameter tuning. Creating an exhaustive list of GridSearchCV is computationally expensive so we chose between two to three values for parameters other than 'solver', and ultimately ended up choosing an alpha value of 0.1, no fit intercept, and a Least-Squares QR solver. Our data for the linear regression model spanned from December 12, 2012 to December 12, 2022. Testing was done on the last 30 days of the data set, which meant that everything before was partitioned as the training set. For the first half of our linear regression model, which was using only one stock's previous closing prices to predict future one, we ran the above brute force algorithm and found that the optimal number of previous days to use was 18 for Apple. Below is a graph of the predicted stock prices in the last 1.5 months of

2022. As we can see, the model fits a small portion of the data well, but has room for improvement due to its MSE of 22.5. The second half of this section involved creating a 'combined' model that

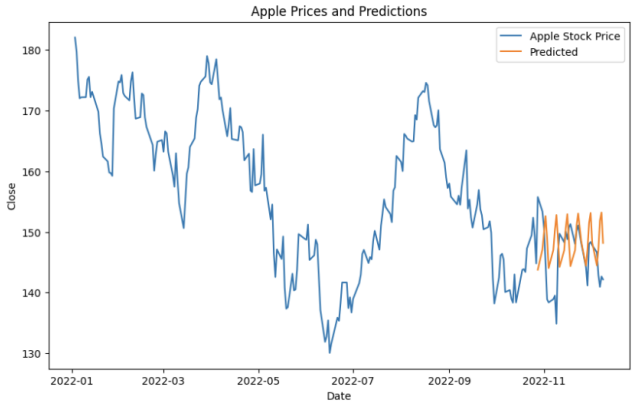

Figure 6: AAPL Prediction vs. Real Price

used previous closing stock prices of multiple companies to use as features in predicting the stock price of one of those multiple companies. This process is then repeated for every single company in the original group. Since we were unsure which companies to group together, we chose to use the companies that make up the FAANG acronym. The same brute force method for choosing the number of previous days was used, and they were 98, 26, 12 and 5 for Amazon, Google, Meta and Netflix respectively. The wide difference between the numbers to use for each company in this 'combined' model raises suspicion because it means that a linear model may not be easily applicable to big tech stocks. A large assumption was made that the best number of days to use for each company would be the best number of days to use in this new model as well. Regardless, a model was trained with the last 30 dates being reserved for testing, and prior dates as the training set. This model also made predictions using its own previous predictions, similar to the previous part. The MSEs were 78, 2232, 263, 196, and 32399 for Apple, Amazon, Google, Meta and Netflix, respectively. The only MSE that improved under this new model was Meta's, which was 603 in the previous model. Even then, there is still clear overfitting, as there is in all the other models. Below are the predictions compared to the actual closing prices of the companies using the 'combined' model. The takeaway from this is that grouping similar companies together to use as features may not be as useful as we hypothesized before, and only add more noise for the majority of FAANG in this case. Further testing will be needed to find the optimal number of features from each company's stock prices, but this approach will most likely be very time consuming and offer less impressive results compared to the other methods discussed here.

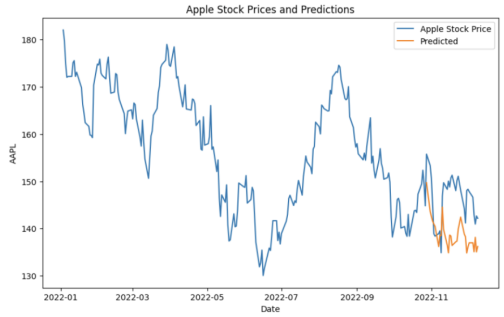

Figure 7: AAPL predictions

Figure 8: AMZN predictions

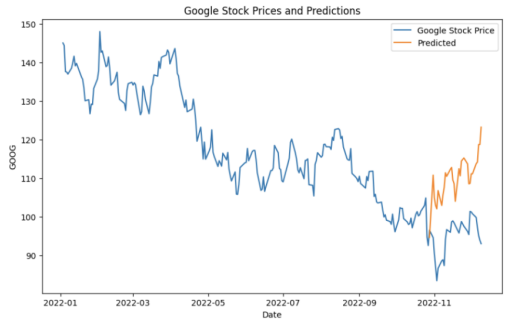

Figure 9: GOOGL predictions

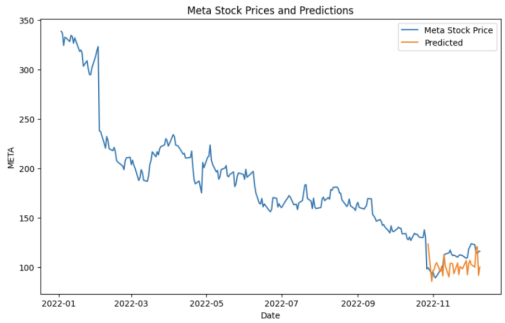

Figure 10: META predictions

6 Random Forests

The Random Forest model took another approach, trying to take advantage of the cyclical nature of stock price movement. The Random Forest model was run two different times using a different set of features for each. The first used historical data of previous days Open, High, Low and Close Price to predict the next day's closing price. Historical data allows the model to gain insight on trends and momentum in the market and stock, and future prices can recognize these movements in making its predictions. Our model takes the average Open, High, Low, etc price of the previous 10 days, but this as well can be adjusted to another metric. The biggest difficulty in training a Random Forest regression model was overfitting on the training data, as the decision trees should not get too hyper specific on the data it is being learned on. After careful experimentation, the model performed best at a depth of 4 with 160 estimators.

The model did not perform greatly on the test data, with a coefficient of determination of 75 percent and a mean-squared error above 29. Looking at the graph, the model fails to predict daily prices at an accurate rate, but it does match the longer term trends. The more nuanced shifts are not captured by this model. In predicting future prices, the model also did not perform very well, as the price predicted by the model differed rather drastically than the actual prices for the day.

Figure 11: Original vs. Predicted Close Price

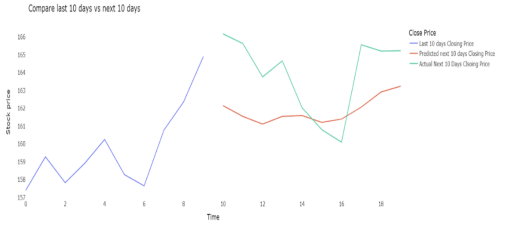

Figure 12: Last 10 Days vs. Next 10 Days

Figure 13: Original vs. Predicted Close Price

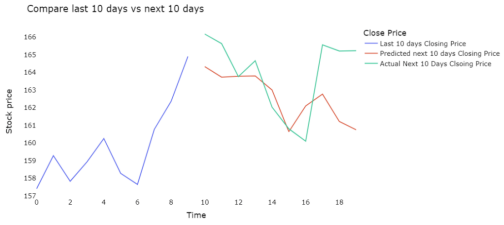

Figure 14: Last 10 Days vs. Next 10 Days

One issue with the prediction of this model was that it had a very limited forecast horizon. The model only outputs a closing price, but uses other features such as volume traded in the previous days when making its prediction. Therefore, if we tried to predict the closing price two weeks in the future, the model does not have information about the volume traded in the days prior to the prediction date, forcing it to use outdated information. Another model was therefore built using only the previous 10 days' closing price as a feature. This model actually performed much better, with a MSE of only 10 and a test score of over 90 percent. Moreover, the prediction horizon could be expanded indefinitely, as it would generate new close prices that it would then use for prediction. The predictions for this model are still not extremely accurate, but they do follow the same movement pattern as the actual stock, which can help with buy/hold/sell decisions that investors make daily.

7 Results

The following table describes the results from the investing strategy described for the LSTM.

	Predicted		Actual	
	Profit/Loss	% Gain/Loss	Profit/Loss	% Gain/Loss
Stacked				
Average	288.96	9.63%		
Max	546.95	18.23%		
Min	-6.7	-0.22%		
Bidirectional				
Average	185.16	6.17%		
Max	400.95	13.37%		
Min	8.58	0.29%		
Dropout + L2				
Average	112.67	3.76%		
Max	224.48	7.48%		
Min	26.95	0.90%		
Additional Features, MACD, RSI				
Average	178.86	5.96%		
Max	409.4	13.65%		
Min	-96.32	-3.21%		
Additional Feature Volume				
Average	159.27	5.31%		
Max	435.97	14.53%		
Min	-101.99	-3.40%		
Actual	-146.11	-4.87%	-146.11	-4.87%

Table 1: Summary of results for various models

8 Conclusion

Our research has highlighted the difficulty in predicting the exact price of a stock for the next day which is a highly challenging task. However, our model has shown promising results in capturing long term trends, accurately predicting sharp price movements, and identifying rising and falling periods with good accuracy. This can be highly valuable for long term trading strategies, such as buy-and-hold, where investors can purchase undervalued stocks that our model identifies and hold onto them as their value increases, ultimately selling them off at their maximum when our model predicts their price will start to decline.

As we look towards the future of this project, we can explore the possibility of integrating our predictions with the Robinhood API to automatically trade stocks based on our model's recom-

mendations. While it may be difficult to make day-to-day predictions due to the volatility of stock prices, our model can identify cyclical patterns in the stock market, allowing for more informed buy, sell, and hold decisions. This would represent a significant advancement in the field of automated stock trading and could potentially lead to higher returns for investors.