COS30018 – Project Summary Report

Project Summary Report

Le Bao Nguyen (104169837)

Table of Contents

I. Introduction.

1. Project Overview.

The goal of the Stock Price Prediction System project is to use machine learning techniques to predict future stock values by using past data. The system was developed and tested in a Google Collab environment using Python as part of the project for the COS30018 - Intelligent Systems course. The system's design, installation, testing, and outcomes are all documented in this paper.

2. Summary of the Features.

The Stock Price Prediction System is designed with the following capabilities:

- Retrieves historical Yahoo Finance stock price data.
- Data is cleaned, scaled, and enhanced in order to get it ready for model training.
- Trains on historical data using a variety of machine learning models, including as LSTM, GRU, and RNN.
- Generates stock price predictions in single, multistep, and multivariate formats.
- Combines predictions from several models to improve prediction accuracy.
- Incorporates other data sources to increase prediction accuracy, including Google Trends.

3. How to run the code.

Using the user-friendliness and computing capacity of Google Collab, the system is run there. To run the Stock Price Prediction System, follow these steps:

1. Open google collab and create new notebook
2. Upload the python file to your Google Collab environment.
3. Ensure that all required libraries are installed.
4. Execute the cells in the python file sequentially. This includes data loading, preprocessing, model training, and prediction generation.
5. Analyze the produced predictions and compare their accuracy to the actual stock prices. The notebook output will include a presentation of the assessment metrics and visuals.
6. Modify and Experiment.

II. Overall System Architecture.

A modular design is used by the Stock Price Prediction System to provide scalability and enable experimentation. Data cleaning, feature scaling,

and engineering are some of the preprocessing activities that come after the initial data gathering from Yahoo Finance. The main process is teaching several machine learning models—such as LSTM, GRU, and RNN—to forecast stock values. Single-step, multistep, and multivariate predictions are supported by the system, and metrics like MSE, MAE, and RMSE are used to assess performance. To increase accuracy, an ensemble learning strategy blends model outputs. The system is set up on Google Collab, which provides an interactive UI for managing and executing experiments.

III. Implemented Data Processing Techniques.

The Stock Price Prediction System uses an extensive set of data processing methods to guarantee high-quality data for machine learning models, improving the accuracy of predictions.

Data collection: Yahoo Finance is the source of historical stock price data; the yfinance library gives access to an array of financial data.

Data cleaning: To preserve the integrity of the dataset, the system fills in the missing data points using interpolation techniques and, if needed, forward or backward filling techniques. Outliers that might affect the model's predictions are found and eliminated using statistical methods and visualization techniques.

Data scaling: Standardization and Min-Max scaling are two methods for scaling features that guarantee that each feature contributes equally to the model training process and enable the neural network models to converge more quickly.

Data normalization: To stabilize neural network learning, time series data is normalized to a specified range.

Feature engineering: Sliding windows are used to turn the dataset into sequences of data points, which are necessary for training sequence models such as GRU and LSTM.

Data reshaping: The information has been modified to fit into the correct format needed to feed machine learning models—particularly sequence models. To do this, the data must be arranged into three-dimensional arrays, the dimensions of which correspond to the number of samples, time steps, and characteristics. For models like as LSTM and GRU, which depend on a certain input shape to efficiently capture temporal relationships, this restructuring is important.

Train-Test split: The time-based splitting of the dataset into training and testing sets ensures that the model is evaluated on new data points that it

hasn't encountered during training. To modify hyperparameters and avoid overfitting, some of the training data is split into a validation set.

IV. Experimented Machine Learning Techniques.

To find the best-performing architecture for stock price prediction, a number of machine learning models were tested:

LSTM (Long Short-Term Memory) model: Multiple layers were included in this model's configuration to capture temporal relationships in the data. Because LSTM networks can handle the vanishing gradient problem and store information over extended periods of time, they are especially well-suited for time series forecasting. To improve the performance of the model, many LSTM layer configurations, such as stacked and bidirectional LSTMs, were examined.

GRU (Gated Recurrent Unit) model: The GRU was examined as a way to lower computational complexity without losing the capacity to record sequential relationships, much like LSTM but with a more straightforward design. When computational finances is an issue, GRUs are frequently chosen over LSTMs because they have shorter training durations and require fewer parameters.

RNN (Recurrent Neural Network) model: To create a baseline performance, a simple recurrent architecture was employed. Traditional RNNs were a helpful guide for improved models like LSTMs and GRUs, despite being unable to handle long-term dependencies because of the vanishing gradient problem.

Arima (AutoRegressive Integrated Moving Average) model: The linear connections within the data have been predicted using ARIMA, a traditional time series forecasting method. To guarantee that the model was able to capture the key patterns in the stock price data, the model's parameters (p, d, and q) were carefully chosen. The dataset's linear trends were shown by ARIMA, which was a useful benchmark even though it was less complex than deep learning models.

Ensemble prediction: Ensemble methods have been used in order to take advantage of the advantages of several models. This featured methods to aggregate predictions from several models, such boosting and bagging, to increase the total prediction accuracy. An ensemble approach was also implemented to combine the predictions from deep learning models and the ARIMA model, aiming to leverage the strengths of both approaches.

V. Developed Extensions.

To enhance prediction accuracy, the system was expanded to include external data sources and ensemble methods of learning in addition to its basic responsibilities. An important improvement was the addition of data from Google Trends.

Google Trends Integration: Google Trends data was added to the algorithm in order to account for more outside variables that may affect stock prices. The method sought to find trends and connections between changes in stock prices and public interest by examining the volume of searches for appropriate keywords. The stock price forecasts have a wider context thanks to this external data source, which may also be able to capture market mood and new patterns that aren't always visible from past stock prices.

VI. Demonstration Scenarios.

1. Parameter Modification:

```
# Example usage
if __name__ == "__main__":
    ticker = 'AMZN'
    start_date = '2016-01-01'
    end_date = '2024-03-20'
    local_file = 'amzn_data.csv'

    # Specify columns to scale
    columns_to_scale = ["Volume"]

    # Load and process data
    train_data, test_data, scalers, original_data, scaled_data = load_and_process_data(
        ticker=ticker,
        start_date=start_date,
        end_date=end_date,
        local_file=local_file,
        split_ratio=0.7,
        columns_to_scale=columns_to_scale
    )

    # Display the original and scaled data tables
    display_custom_table(scaled_data, num_rows=5)

    # Load trend data separately and display it
    trends_cache_file = f"/content/drive/My Drive/Cos30018/{ticker}_trends.csv"
    trends_data = pd.read_csv(trends_cache_file, index_col='date', parse_dates=True)
    display_trend_data(trends_data)

    # Task 3 - Data Processing 2 (code to run)

    # Plot candlestick chart
    candlestick_chart(original_data[(original_data.index >= start_date) & (original_data.index <= end_date)], title=f'{ticker} Candlestick Chart', n_days=7, candle_width=5)  # Use larger n_days to make each candlestick larger

    # Display the boxplot chart with volume
    boxplot_chart(original_data, title=f'{ticker} Boxplot and Volume Chart', window_size=20)
```

Figure 1: The parameter to run the code (1).

```
    # Display the boxplot chart with volume
    boxplot_chart(original_data, title=f'{ticker} Boxplot and Volume Chart', window_size=20)

    # Task 4 - Machine learning 1 (code to run)

    lstm_config = {
        "layers": [
            {"type": "LSTM", "units": 50, "return_sequences": True},
            {"type": "LSTM", "units": 50, "return_sequences": False},
            {"type": "Dense", "units": 1}
        ]
    }

    gru_config = {
        "layers": [
            {"type": "GRU", "units": 50, "return_sequences": True},
            {"type": "GRU", "units": 50, "return_sequences": False},
            {"type": "Dense", "units": 1}
        ]
    }

    rnn_config = {
        "layers": [
            {"type": "RNN", "units": 50, "return_sequences": True},
            {"type": "RNN", "units": 50, "return_sequences": False},
            {"type": "Dense", "units": 1}
        ]
    }

    layers_configs = [lstm_config, gru_config, rnn_config]

    results = experiment_with_models(train_data, test_data, scalers["Close"], layers_configs, epochs=50, batch_size=32)

    # Task 5 - Machine learning 2 (code to run)
    key_value = 6 # You can change this to any number of future steps you want to predict
```

Figure 2: The parameter to run the code (2).

```
[ ]     # Multistep prediction example
        best_model = results[0]["history"].model  # Using the first model as an example
        multistep_preds = multistep_prediction(best_model, test_data, scalers["Close"], k=key_value)
        print(f"Multistep Predictions for next {key_value} days:", multistep_preds)

        # Multivariate prediction example
        multivariate_preds = multivariate_prediction(results[0]['model'], test_data, scalers["Close"], feature_scalers=scalers, future_day=1)
        print("Multivariate Prediction for the next day:", multivariate_preds)

        # Multistep multivariate prediction example
        multistep_multivariate_preds = multistep_multivariate_prediction(best_model, test_data, scalers["Close"], k=key_value)
        print(f"Multistep Multivariate Predictions for next {key_value} days:", multistep_multivariate_preds)



        # Task 6 - Machine Learning 3 (Code to run)
        print("Fitting ARIMA model...")
        best_dl_model = results[0]  # Assuming the first model is the best performing for simplicity
        dl_predictions = best_dl_model["predicted_stock_price"]
        real_stock_price = best_dl_model["real_stock_price"]

        arima_predictions = fit_arima_model(train_data, test_data, scalers['Close'], order=(5, 1, 0))
        arima_predictions = arima_predictions[:len(real_stock_price)]  # Trim ARIMA predictions to match the length

        ensemble_preds = ensemble_predictions(dl_predictions, arima_predictions, weights=[0.7, 0.3])

        plt.figure(figsize=(14, 5))
        plt.plot(real_stock_price, color='red', label='Real Stock Price')
        plt.plot(dl_predictions, color='blue', label='LTSM Predictions')
        plt.plot(arima_predictions, color='green', label='ARIMA Predictions')
        plt.plot(ensemble_preds, color='purple', label='Ensemble Predictions')
        plt.title('Stock Price Prediction: LTSM vs ARIMA vs Ensemble')
        plt.xlabel('Time')
        plt.ylabel('Stock Price')
        plt.legend()
        plt.show()
```

Figure 3: The parameter to run the code (3).

```
# Task 7 - Extension (Code to run)
arima_predictions = fit_arima_model(train_data, test_data, scalers['Close'])

ensemble_preds = ensemble_predictions(results[0]['predicted_stock_price'], arima_predictions)
plt.figure(figsize=(14, 5))
plt.plot(scalers['Close'].inverse_transform(test_data), color='red', label='Real Stock Price')
plt.plot(ensemble_preds, color='green', label='Ensemble Predicted Stock Price (With Google trends)')
plt.title(f'Ensemble Stock Price Prediction (With Google trends data)')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()

mse = mean_squared_error(scalers['Close'].inverse_transform(test_data), ensemble_preds)
print(f"Mean Squared Error of the Ensemble Model: {mse}")
```

Figure 4: The parameter to run the code (4).

Throughout the experimental phase, a number of hyperparameters were carefully adjusted to guarantee the machine learning models operated at their best. To determine the optimal combinations, a combination of grid search, domain expertise, and trial & error was used. The number of units, batch size, number of epochs, and model architecture are the main hyperparameters that were changed.

Number of Units: After testing with various sizes, set at 50 for the LSTM, GRU, and RNN models, balancing temporal dependency capture and complexity.

Batch Size: Set at 32, offering an appropriate balance between model convergence and training speed.

Number of Epochs: Fixed at 50, verified by observing validation loss and training, to prevent both underfitting and overfitting.

Model Architecture:

- **LSTM and GRU:** Two layers with 50 units each and a Dense layer with 1 unit.
- **RNN:** Similar structure with two RNN layers and a Dense layer.

ARIMA Model: Configured with order=(5, 1, 0), chosen for its ability to capture autocorrelations in the data.

Ensemble Learning: Combined deep learning models and ARIMA predictions with weights of 0.7 and 0.3, respectively, to minimize the mean squared error.

2. Charts:

Figure 5: The candlestick chart of AMZN data.

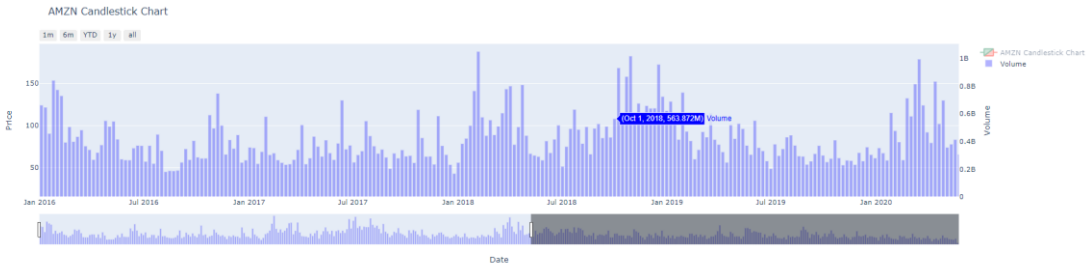

Figure 6: The volume chart of AMZN data.

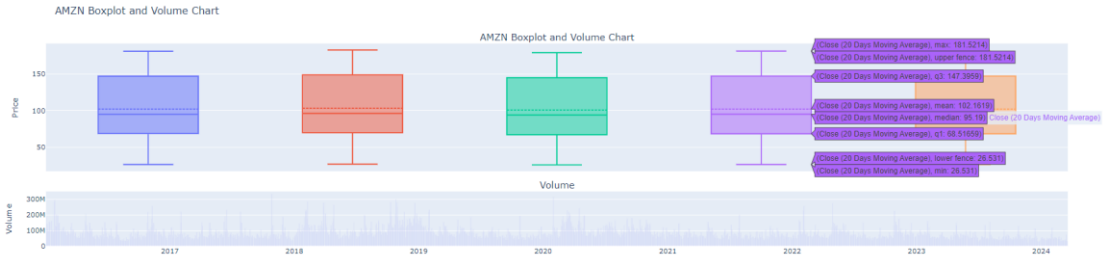

Figure 7: The boxplot chart of AMZN data.

Candlestick Chart: The open, high, low, and close prices for each day are displayed on this chart, which was created to illustrate the stock's price movement over time. Each candlestick was made more noticeable by increasing the n_days parameter, which improved the visibility of the patterns and trends in the movements of the stock price.

Boxplot Chart: To show the distribution and changes of stock prices over certain time periods, boxplot charts were made. These charts, which showed the median, quartiles, and possible outliers using window widths of 20 and 40 days, provided details about the volatility and price range of the stock during those times.

3. Predictions:

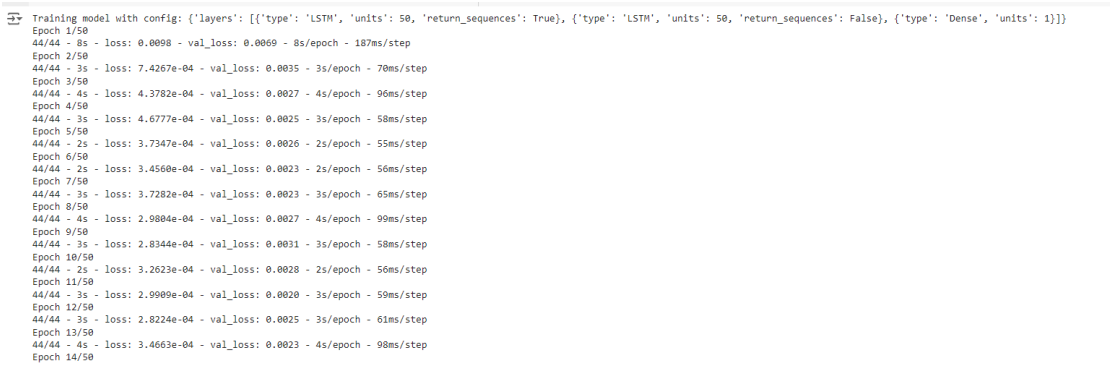

Figure 8: The epoch to train and test data.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
Multistep Predictions for next 5 days: [[172.36057]
 [170.94423]
 [169.26804]
 [167.5212 ]
 [165.77399]
 [164.07582]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
Multivariate Prediction for the next day: [[172.36057]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
Multistep Multivariate Predictions for next 5 days: [[1.7236057e+02]
 [6.1200889e+03]
 [2.3880991e+05]
 [9.3088230e+06]
 [3.6285594e+08]
 [1.4144043e+10]]
```

Figure 9: The multistep, multivariate and the combination of both predictions.

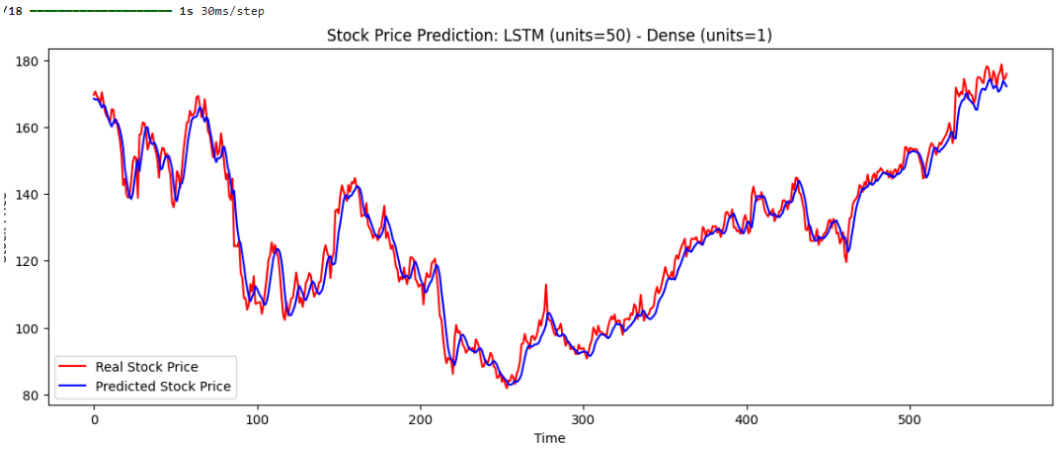

Figure 10: The stock prediction with LSTM model.

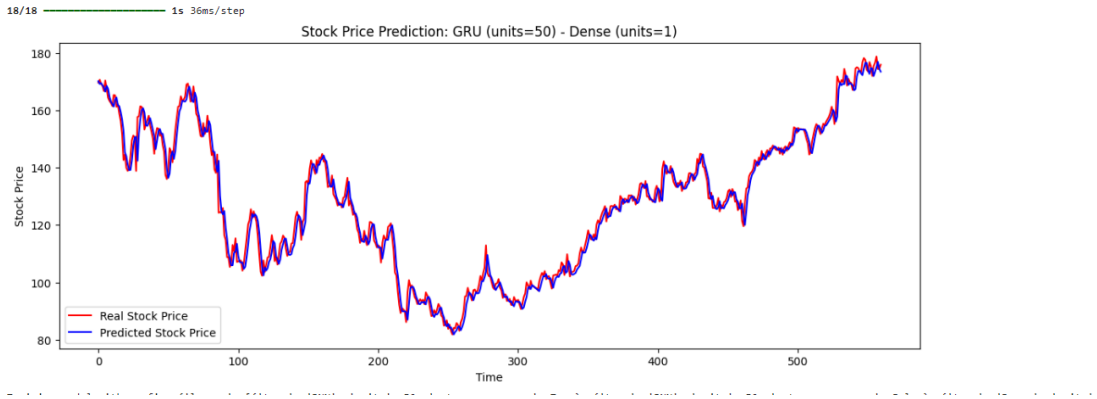

Figure 11: The stock prediction with GRU model.

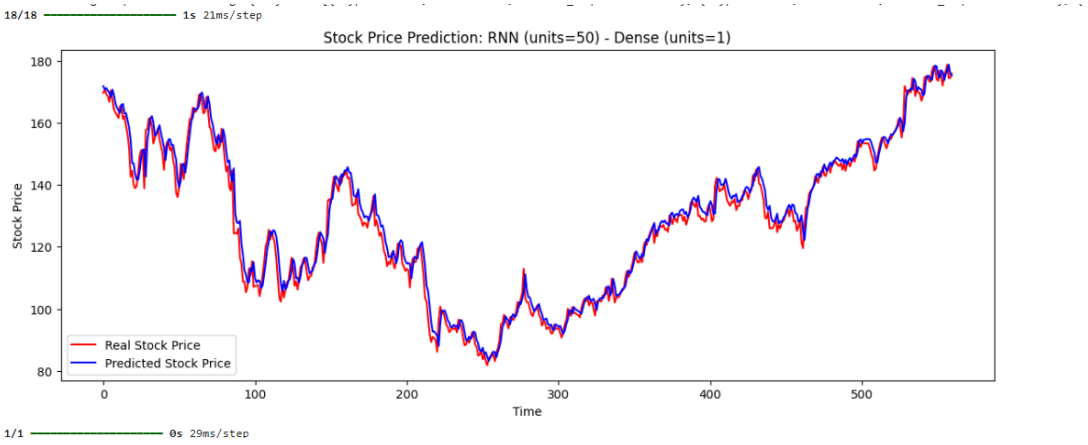

Figure 12: The stock prediction with RNN model.

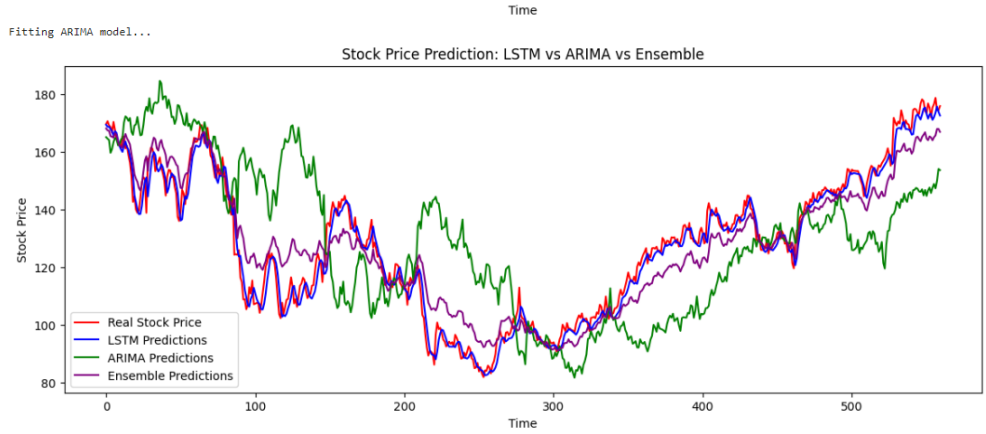

Figure 13: The Arima and ensemble stock prediction.

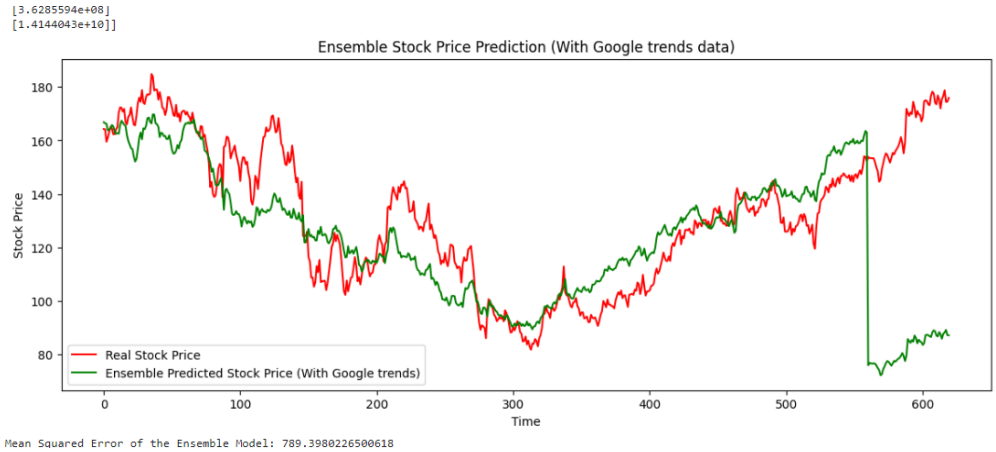

Figure 14: The stock prediction (with Google trends).

Multiple methods and visualizations were used in order evaluate the models and prediction stock prices:

- **LSTM, GRU, and RNN Predictions:** We generated and compared predictions from the LSTM, GRU, and RNN models. The scaled data was used for training and testing the models, and the accuracy of their predictions was used to evaluate how well they captured the patterns in stock prices.
- **Arima Model Predictions:** An additional set of predictions have been generated by fitting an ARIMA model to the stock price data. The predictions were compared with those produced by deep learning algorithms and actual stock prices.
- **Ensemble Predictions:** The ARIMA model and the deep learning models' predictions were integrated using an ensemble approach. Using the ensemble predictions (0.3 for ARIMA and 0.7 for deep learning models) reduced the mean squared error and increased accuracy.

VII. Critical Analysis.

1. Model Selection and Performance:

The selection of LSTM, GRU, and RNN models made it possible to successfully capture temporal relationships in stock price data. Because LSTM can handle long-term dependencies, it performed the best out of all of them.

GRU is a good substitute in situations where resources are limited because of its more straightforward design, which produced competitive results while reducing computing complexity.

Compared to LSTM and GRU, the basic RNN model was not as effective in capturing complicated temporal patterns, however it was still an effective starting point.

2. Hyperparameter Tuning:

In order to maximize model performance, it was essential to modify hyperparameters such the number of units, batch size, and epochs. It was discovered through experimentation that in order to avoid both overfitting and underfitting, these factors needed to be balanced.

3. Integration of Arima Model:

The prediction system gained an ordinary statistical viewpoint by integrating the ARIMA model. The deep learning models were better at capturing non-linear patterns, and ARIMA's ability in modeling linear correlations was a nice complement to them.

Comparing ARIMA to neural network models, however, showed that ARIMA was less successful in predicting rapid swings in the market due to the non-stationary characteristics of stock prices.

4. Ensemble Method:

The ensemble technique, which combines predictions from the ARIMA model with deep learning, turned shown to be a reliable strategy. The ensemble forecasts improved in accuracy and decreased in prediction error by utilizing the advantages of both model types.

This hybrid method showed that by resolving the shortcomings of each model separately, combining models may often outperform individual models.

5. Google Trends Integration:

The feature set was enhanced by the addition of Google Trends data, which may have improved prediction accuracy. Additional information on market sentiment and outside variables affecting stock prices was supplied by trends data.

However, careful management was needed throughout the integration and preparation of this data to prevent noise introduction, highlighting the need for more research into the most effective use of auxiliary data sources.

VIII. Conclusion.

In summary, the research effectively used statistical models and deep learning to create a reliable stock price prediction system. The ability to combine several models to increase forecast accuracy was shown by the ensemble approach. To improve the prediction power of the system, future research could study the addition of other external data sources and further enhance the ensemble technique.