HUMBER

Faculty of Applied Sciences & Technology

ASSIGNMENT 3 (MILESTONE 1,2)

Google Stock Price Analysis and Prediction using Machine Learning

Anjana Sebastian
Faculty of Business
Humber Institute of Technology and Advanced Learning
Toronto, Ontario
n01510800@humbermail.ca

Chenguiling Feng
Faculty of Business
Humber Institute of Technology and Advanced Learning
Toronto, Ontario
n04169402@humbermail.ca

Jwaleena Jose
Faculty of Business
Humber Institute of Technology and Advanced Learning
Toronto, Ontario
n01510696@humbermail.ca

Rohit Kumar
Faculty of Business
Humber Institute of Technology and Advanced Learning
Toronto, Ontario
n01512529@humbermail.ca

Shoobham Hans
Faculty of Business
Humber Institute of Technology and Advanced Learning
Toronto, Ontario
n01480686@humbermail.ca

***Abstract*— This paper presents a data product for stock price analysis and prediction using machine learning techniques. The goal of the data product is to analyze historical stock market data of Google and forecast its future prices. The paper outlines the problem statement, key requirements for the data product, and the choice of forecasting models, including Prophet and LSTM. The data collection process involves obtaining reliable and up-to-date historical stock market data of Google. Data preprocessing and cleaning techniques are applied to handle missing values and outliers. Time series analysis is performed to identify trends, seasonality, and patterns in the data. The forecasting models are trained and evaluated using performance metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE). The data product provides visualizations to display historical stock prices, trends, and forecasted values, making it user-friendly for investors and businesses in the stock market domain. The success of the data product is measured based on the accuracy of the forecasting models and positive user feedback about its usability and effectiveness.**

Keywords*— *Stock Price Analysis and Prediction, Machine Learning, Time Series Analysis, Forecasting Models, Prophet, LSTM, Data Preprocessing, Data Cleaning, Performance Metrics, Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), Data Visualization, Stock Market Domain.

MILESTONE 1

I. PROBLEM STATEMENT

The analytics problem is to develop a data product that analyzes historical stock market data of Google and predicts its future prices using various forecasting models. The goal is to achieve accurate predictions and provide valuable insights to investors and businesses in the stock market domain.

The data product needs to have access to a reliable and up-to-date dataset of Google's historical stock market prices. This data will serve as the foundation for analysis and prediction. The data should be preprocessed and cleaned to handle missing values, outliers, and ensure consistency in the time series and it should be capable of performing time series analysis on the historical stock market data to identify trends, seasonality, and other patterns. Moreover, the solution should include forecasting models that can predict future stock prices based on historical patterns. These models can include methods like Prophet, LSTM. Then the forecasting models should be evaluated using appropriate performance metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE). Furthermore, the data product should provide visualizations to display historical stock prices, trends, and forecasted values, making it easier for users to interpret the results.

The proposed data product has the potential to offer significant benefits to both individual investors and businesses operating in the stock market domain. The data product will enable investors and traders to make more informed decisions about buying, selling, or holding Google's stock, leading to potentially higher returns on investments. By providing reliable predictions, the data product can also help investors manage risks and minimize potential losses. The automated forecasting models will save time for investors, eliminating the need for manual analysis and the visualization capabilities will provide valuable market insights to identify patterns and trends. Businesses in the stock market domain can gain a competitive advantage by leveraging the data product's accurate predictions to make strategic decisions. Improved decision-making and risk management can result in increased profitability for individual investors and businesses [1].

The success of the data product will be measured based on the accuracy of the forecasting models. The RMSE and MAPE metrics will be used to evaluate the models' performance. A successful outcome will be achieved if the data product can consistently provide accurate predictions, leading to informed decision-making and increased profitability for users. Additionally, positive feedback from users about the usability and effectiveness of the data product will also indicate its success.

II. CHOICE OF MODEL

To develop the data product for predicting Google's future stock prices, we need to identify the relevant analytical and statistical methods and outline the data science pipeline. So, in data collection we need to obtain access to reliable and up-to-date historical stock market data of Google's stock prices. We need to ensure the dataset includes relevant features like date, opening price, closing price, high, low, Average volume, etc. Then In Data Preprocessing and Cleaning, we need to Handle missing values, if any, through imputation methods such as interpolation or forward/backward filling. Detect and handle outliers that may impact the forecasting models and ensure consistency in the time series by resampling or interpolating if necessary. Then we need to Identify trends, seasonality, and other patterns that could influence the forecasting models. Create visualizations to better understand the data, for example, line plots, box plots. We can also apply time series analysis techniques to identify and account for seasonality, trend, and autocorrelation in the data. In Model Selection, consider various forecasting models suitable for time series data, such as a powerful time series forecasting model that can handle seasonality, holidays, and trend changes. The Long Short-Term Memory (LSTM) neural network, which is a type of deep learning model known for its ability to capture long-term dependencies in time series data.

In Model Training and Evaluation, we need to split the dataset into training and testing sets where we train the selected forecasting models on the training set and evaluate the models using appropriate performance metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE) on the testing set. Then we need to do Forecasting and Visualization, use the trained forecasting models to predict future stock prices. Visualize the historical stock prices, trends, and forecasted values using interactive plots and charts. Provide users with clear visualizations to help them interpret the results effectively.

We can create a user-friendly interface where users can input their desired prediction horizon and view the forecasted stock prices along with the historical data and visualizations. And ensure that the data product is accessible and easy to use for both individual investors and businesses in the stock market domain. Continuously monitor the performance of the forecasting models and update them regularly to account for new data and changing market conditions also, gather user feedback to identify areas for improvement and make necessary updates to enhance the data product's usability and effectiveness.

By following this data science pipeline, the practitioner can create a robust data product that analyzes historical stock market data of Google, provides accurate predictions using various forecasting models, and offers valuable insights to investors and businesses in the stock market domain. The success of the data product will be measured based on the accuracy of the forecasting models and positive feedback from users about its usability and effectiveness.

III. DATA COLLECTION

The dataset we chose is the Google Dataset, which is a dataset of Google for Stock Data Analysis and Price Prediction. This data is from 2014 till December 2022. With the help of this Dataset, one can predict future Stock Values.

The Data Source is:

Google Stock Data 2014-2022. (2022, December 23). Kaggle. https://www.kaggle.com/datasets/malayvyas/google-stock-data

The Features of the Dataset:

- Month Starting: This column represents the starting date of each month for which the stock market data is recorded.
- Open: The "Open" column represents the opening price of Google's stock on the specified date. The opening price is the price at which the stock's trading begins at the beginning of the trading day.
- High: The "High" column represents the highest price reached by Google's stock during the trading day on the specified date.
- Low: The "Low" column represents the lowest price reached by Google's stock during the trading day on the specified date.
- Close: The "Close" column represents the closing price of Google's stock on the specified date. The closing price is the price at which the stock's trading ends at the end of the trading day.
- Change %: This column represents the percentage change in the stock price from the previous trading day's closing price to the current trading day's closing price where a positive value shows that there is an increase in the stock price where the negative value indicates a decrease in the stock price.
- Avg. Volume: The "Avg. Volume" column represents the average trading volume of Google's stock for the specified month. [2].

This dataset provides historical stock market data for Google's stock prices over several months. Each row in the dataset represents the stock market data for a specific month, including the opening, highest, lowest, and closing prices, as well as the percentage change in price and the average trading volume for that month. This information can be used to perform time series analysis, identify trends and patterns, and develop forecasting models to predict future stock prices.

MILESTONE 2

We import various libraries and modules necessary for data manipulation, statistical modeling, time series analysis, and data visualization. After that, we read the data from the Excel file using Pandas, converting it into a DataFrame. The dataset has been checked for null values, and no missing values are present, and we use the code to reset the columns based on years.

This is a Time series dataset, and we don't have any missing values, duplicate data's, Irrelevant and Incorrect Data and no categorical data.

Then we are displaying the correlation matrix of the stock dataset. A correlation of 1 indicates a positive correlation, a correlation of -1 indicates a negative correlation, and a correlation close to 0 indicates no linear relationship between the variables. Here from our dataset Open, High, Low, and Close these columns have strong positive correlations with each other and In Change %, this column has weak positive correlation values with Close and a weak negative correlation with Open, High, and Low. This shows that there might be a slight relationship between daily percentage changes in stock prices and the opening, highest, and lowest prices. And the Avg. Volume column has weak negative correlations with all other columns. This indicates that there might be a slight inverse relationship between the average trading volume and stock prices.

```
In [8]: correlation_matrix = Google_df.corr()

        # Displaying  the correlation matrix of the stock dataset
        print(correlation_matrix)
```

```
                  Open      High       Low     Close  Change %  Avg. Volume
Open          1.000000  0.996097  0.993130  0.988203 -0.097338    -0.372177
High          0.996097  1.000000  0.994040  0.994515 -0.031083    -0.362881
Low           0.993130  0.994040  1.000000  0.995412 -0.017648    -0.417914
Close         0.988203  0.994515  0.995412  1.000000  0.038501    -0.392309
Change %     -0.097338 -0.031083 -0.017648  0.038501  1.000000    -0.144140
Avg. Volume  -0.372177 -0.362881 -0.417914 -0.392309 -0.144140     1.000000
```

Fig 1.1

```
In [9]: plt.figure(figsize=(10, 8))  # Optional: Adjust the figure size
        sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
        plt.title('Correlation Matrix Heatmap')
        plt.show()
```

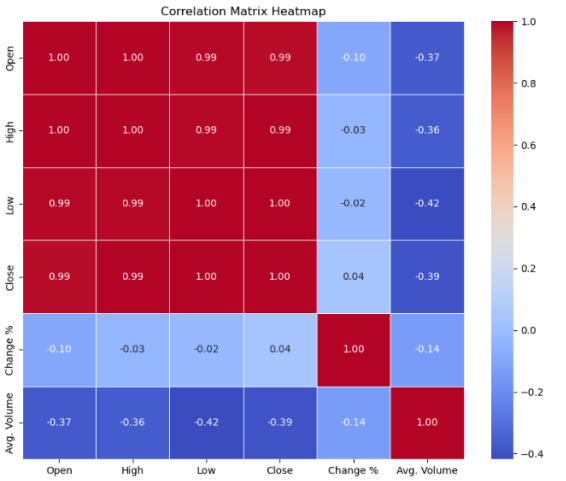

Fig 1.2

After the correlation matrix we find the outliers. From figure 1.3 it shows that the Average volume has some outliers. But here we are not using the average volume for our forecasting so we are not considering this outliers for further analysis.

```
In [10]: numeric_columns = Google_df.select_dtypes(include=[np.number])

         # Create box plots for each numeric column
         plt.figure(figsize=(12, 8))
         sns.boxplot(data=numeric_columns)
         plt.title('outliers detection')
         plt.xticks(rotation=45)
         plt.show()
```

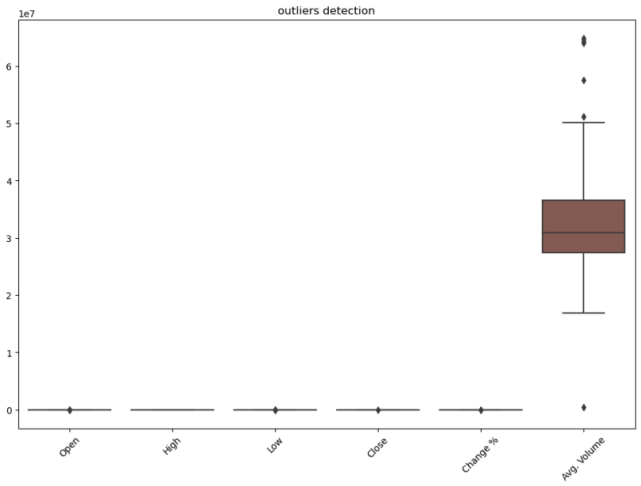

Fig 1.3

Then we convert the column Month Starting to datetime format. After converting to datetime format, we calculated the number of null values and displayed them and converting them to actual values.

Then we use a code to check for missing values (null values) and the output indicates that there is one column with missing values in Google_df. That is the "Month Starting" which has 3 missing values. This means that there are three instances where the "Month Starting" data is not available or is represented as a null value in the DataFrame. And we display the rows in the DataFrame Google_df where the "Month Starting" column contains NaT (Not a Time) values. (Fig 1.4) In these rows, the "Month Starting" column contains NaT values (shown as "NaT"), while the other columns have numerical or float values representing stock market data.

```
In [13]: Google_df.isnull().sum()[Google_df.isnull().sum()>0]
Out[13]: Month Starting    3
         dtype: int64

In [14]: # Find the rows with NaT values in the "Month Starting" column
         rows_with_nat = Google_df[Google_df['Month Starting'].isnull()]

         # Display the rows where NaT values occur
         print(rows_with_nat)
```

	Month Starting	Open	High	Low	Close	Change %	Avg. Volume
50	NaT	50.68	55.54	50.31	54.25	0.0665	28953815
62	NaT	59.40	59.54	55.01	55.18	-0.0714	30294330
74	NaT	66.43	72.05	64.95	71.45	0.0595	31890974

Fig 1.4

Next, we are showing different graphs to show the close prices over time to identify the trend (Fig 1.5) where it visualizes how the closing price of Google's stock changes with time, which can help identify the trend in the stock's value. The graph shows the trend of Google's stock price movement over the given period. The other graph plotted are "Close" and "Open" prices of Google's stock over time (Fig 1.6) , the highest price ("High" price) of Google's stock over time (Fig 1.7) , the "High" and "Low" prices of Google's stock over time on the same graph (Fig 1.8) It allows us to compare and visualize how the highest and lowest prices of Google's stock change with time and identify any trends or patterns in their movements. The output graph will show two lines representing the "High" and "Low" prices of Google's stock over time. By comparing the two lines, we can observe their movements and identify any trends or patterns in the stock's price behavior. And the average volume ("Avg. Volume") of Google's stock over time (Fig 1.9).

The graph shows how the closing price of Google's stock changes with time, which can help identify the trend in the stock's value. The graph shows the trend of Google's stock price movement over the given period. Where the google close price over time, close and open price over time, high price over time, and google high and low price over time shows an Increasing trend from year 2014 to 2021 where it slightly started decline in between the year 2021 and 2022.

```
In [20]: #Plotting close price over time to identify the trend.
         plt.figure(figsize=(15, 5))
         plt.plot(Google_df["Month Starting"], Google_df["Close"],marker='o', linestyle='-', color='b')
         plt.xlabel("Year")
         plt.ylabel("Close")
         plt.title("Google Close Price Over Time")
         plt.xticks(rotation=45)
         plt.tight_layout()
         plt.show()
```

Fig 1.5

```
# Plotting the "Close" and "Open" prices over time
plt.figure(figsize=(12, 6))
plt.plot(Google_df['Month Starting'], Google_df['Close'], label='Close', color='r', linestyle='dashed')
plt.plot(Google_df['Month Starting'], Google_df['Open'], label='Open',color='orange', linestyle='dashed' )
plt.xlabel('Month Starting')
plt.ylabel('Price')
plt.title('Google Close and Open Prices Over Time')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

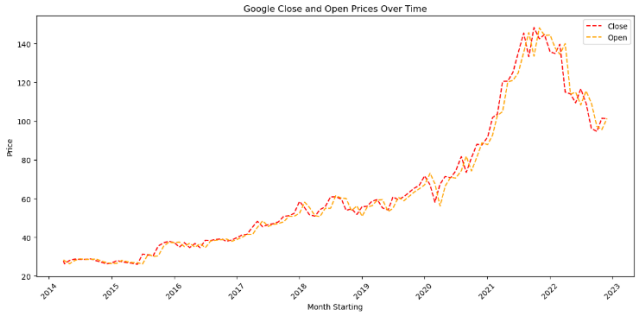

Fig 1.6

```
#Plot the pattern of highest price over time.
plt.figure(figsize=(12, 6))
plt.plot(Google_df["Month Starting"], Google_df["High"], marker='o', linestyle='-', color='b')
plt.xlabel("Year")
plt.ylabel("High")
plt.title("Google High Price Over Time")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Fig 1.7

```
# Plot the "High" and "Low" prices over time
plt.figure(figsize=(12, 6))
plt.plot(Google_df['Month Starting'], Google_df['High'], label='High', color='r', linestyle='dashed')
plt.plot(Google_df['Month Starting'], Google_df['Low'], label='Low', color='orange', linestyle='dashed')
plt.xlabel('Month Starting')
plt.ylabel('Price')
plt.title('Google High and Low Prices Over Time')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

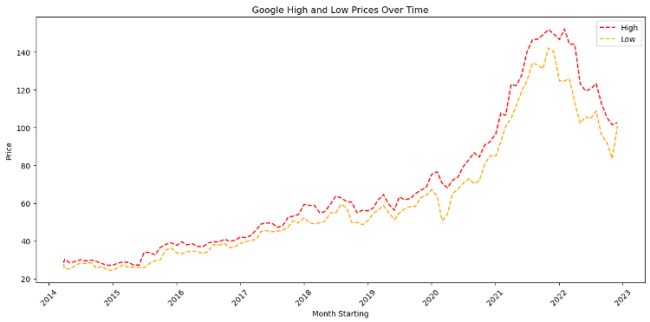

Fig 1.8

```
plt.figure(figsize=(12, 6))
plt.plot(Google_df["Month Starting"], Google_df["Avg. Volume"],marker='o',linestyle='-', color='b')
plt.xlabel("Year")
plt.ylabel("Avg.Volume")
plt.title("Google Average Volume Over Time")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

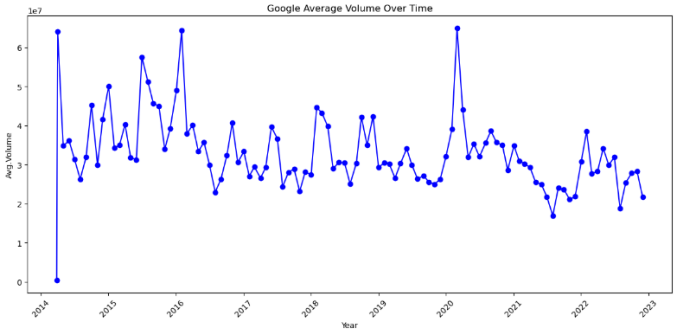

Fig 1.9

The google average volume over time is fluctuation over time. Where the peak time is 2014, 2016 and 2020 approximately, which had the higher trading activity in Google's stock over time.

Next is the scattered plot (Fig 2.0)

```
plt.figure(figsize=(12, 6))
plt.scatter(Google_df['Avg. Volume'], Google_df['Close'], marker='o', color='b', alpha=0.5)
plt.xlabel('Average Volume')
plt.ylabel('Close Price')
plt.title('Effect of Average Volume on Google Stock Price')
plt.grid(True)
plt.tight_layout()
plt.show()
```

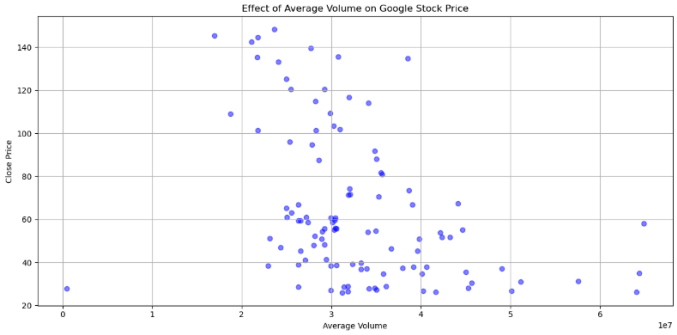

Fig 2.0

This graph shows a downward-sloping pattern from left to right, it indicates a negative correlation between volume and price. This means that as the average volume increases, the closing price tends to decrease.

Next is Centered and Trailing moving averages.

```
# Calculate the centered moving average (window size = 3)
centered_ma = close_ts.rolling(window=3, center=True).mean()

# Calculate the trailing moving average (window size = 3)
trailing_ma = close_ts.rolling(window=3).mean()

# Plot the original time series, centered moving average, and trailing moving average
plt.figure(figsize=(12, 6))
plt.plot(close_ts.index, close_ts, label='Close', color='b')
plt.plot(centered_ma.index, centered_ma, label='Centered Moving Avg.', color='r', linestyle='dashed')
plt.plot(trailing_ma.index, trailing_ma, label='Trailing Moving Avg.', color='g', linestyle='dashed')
plt.xlabel('Month Starting')
plt.ylabel('Close Price')
plt.title('Google Closing Price Time Series with Moving Averages')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

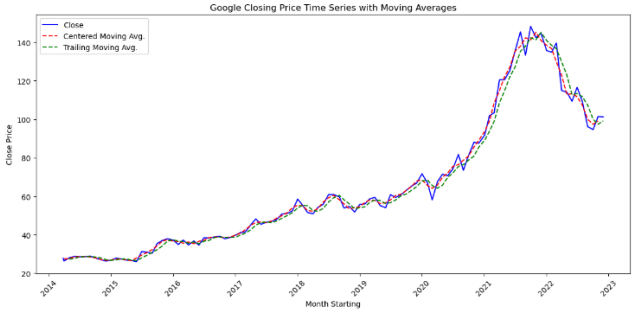

Fig 2.1

Here we plot two types of moving averages, the Centered Moving Average and the Trailing Moving Average, for Google's closing stock price time series. It also shows an Increasing trend from 2014 to 2021 and again slightly decreased between the year 2021 and 2022.

We changed the y-axis limits depending on the supplied ylim and the x-axis limits to span the time period from 2014 to 2022. For the training and validation phases, it adds lines and text annotations to visually separate them. Then, to create the layout for two subplots, we defined the function graph Layout (axes, train_df, valid_df)..

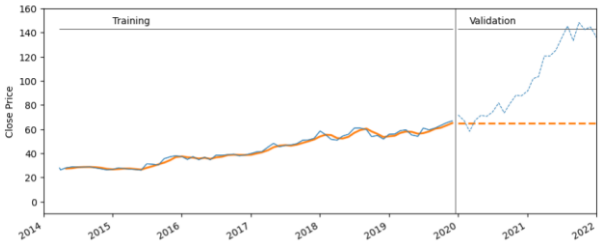

Fig 2.2

The layout of the two subplots is then set up by calling the single Graph Layout function twice, passing the training and validation data frames (train_df and valid_df) and the two subplots (axes). It divides the data into training and validation sets, with valid_ts containing the final 36 data points of the close_ts time series and train_ts containing the first (len(close_ts) - 36) data points. The moving average was then computed using a rolling window of size 3 on the training data and stored in the ma_trailing variable. With a smaller window size, the model can capture short-term fluctuations and

respond quickly to changes in the data. It can be useful in identifying short-term trends and turning points. By taking the most recent moving average value from the training data to construct the moving average prediction for the validation time range, it generates a forecast for the validation period. Additionally, the first subplot (axes[0]) depicts the time series, the moving average on the training data, and the forecast. By deducting the moving average prediction from the actual validation data, it determines the forecast errors (residuals), and then plots those residuals on the second subplot (axes[1]). Finally, we used the plt.tight_layout() and plt.show() functions to display the plots after setting up the layout for the subplots and modifying labels and legends. The code's overall goal is to plot on two subplots the time series data, moving average, and forecast errors for the validation period. Knowing how well the moving average forecast performs in comparison to the actual validation data is useful.

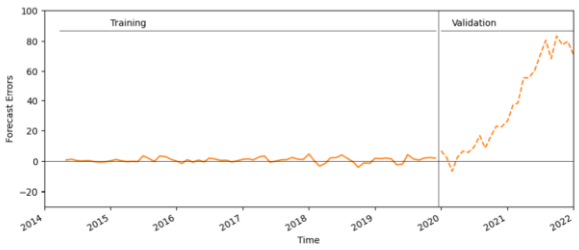

Fig 2.3

Then, we sought to separate a time series into its trend, seasonality, and residual components. The decomposition is carried out using the seasonal_decompose function from the statsmodels.tsa.seasonal library. To see how each component contributed to the original time series individually, the components are then plotted in distinct subplots. The seasonal component displays repeating patterns over a specific period, which in this example is believed to be 12 months, while the residual component comprises random fluctuations or noise. The trend component indicates the long-term pattern. Understanding the underlying patterns and fluctuations in the initial time series data is aided by this breakdown.

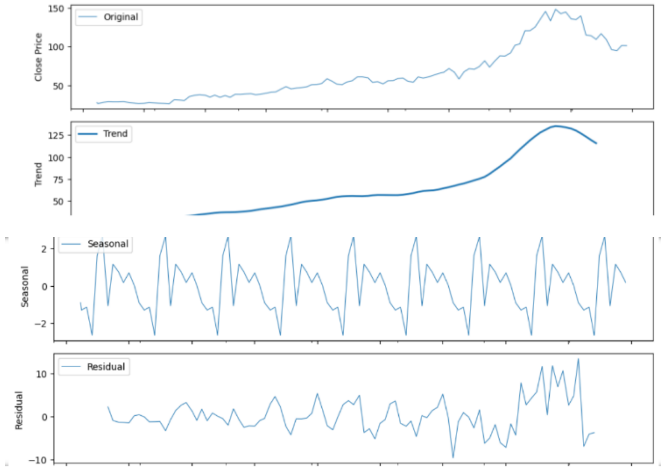

Fig 2.4

After that, we used exponential smoothing to forecast time series. It updates the index of the time series close_ts with dates that are valid as of January 1, 2014. The time series is then divided into a training set (which contains 60% of the data) and a test set (40% of the data). With additive trend, seasonality, and exponential smoothing (seasonal_periods=5), the model is fitted to the training data. On the test set, it then offers predictions. Using matplotlib, the real, trained, and predicted values are each shown in a separate color. This image allows for a comparison of the forecast to the actual test data by demonstrating how effectively the Exponential Smoothing model predicts future values based on the training data.

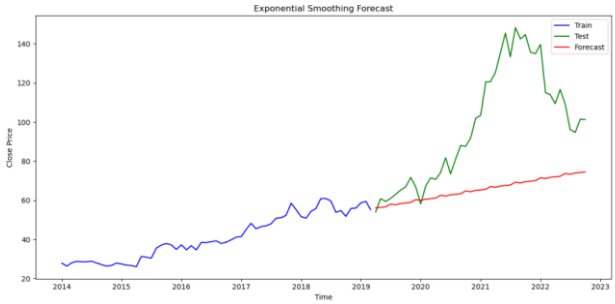

Fig 2.5

Using additive trend and additive seasonal components, we used exponential smoothing to forecast time series. With a 12-month seasonal period, it fits the Exponential Smoothing model to the close_ts time series data. From the final data point in 2022, the code then creates a forecast for the following 12 months. Using matplotlib, the predicted values are presented next to the initial time series data. This graphic offers insights into probable future price movements by demonstrating how well the Exponential Smoothing model forecasts the trend and seasonality in the time series.

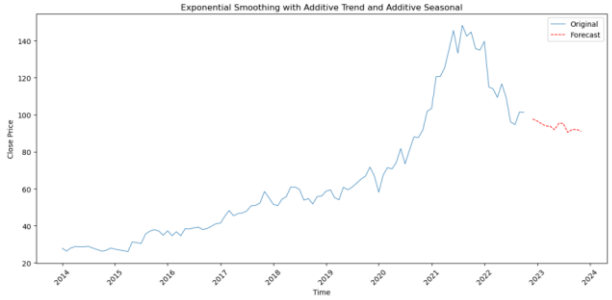

Fig 2.6

Additionally, we used a Long Short-Term Memory (LSTM) neural network to achieve time series forecasting. The close_ts data is normalized using Min-Max scaling. For the purpose of training the LSTM, input data and corresponding output data sequences are constructed with a predetermined sequence length (seq_length). The LSTM model with one LSTM layer and one dense layer is built and trained using mean squared error loss on the training set of data. On the basis of the test data, predictions are made using the trained model. Inverse scaling is then used to change the anticipated values back to the original scale. Finally, a comparison between the forecast and the actual test data is possible thanks to the plotting of the actual and projected values, which shows how well the LSTM performs forecasting.

```
Epoch 1/100
4/4 [==============================] - 2s 8ms/step - loss: 0.0237
Epoch 2/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0194
Epoch 3/100
4/4 [==============================] - 0s 4ms/step - loss: 0.0159
Epoch 4/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0130
Epoch 5/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0104
Epoch 6/100
4/4 [==============================] - 0s 4ms/step - loss: 0.0085
Epoch 7/100
4/4 [==============================] - 0s 6ms/step - loss: 0.0069
Epoch 8/100
4/4 [==============================] - 0s 6ms/step - loss: 0.0059
Epoch 9/100
4/4 [==============================] - 0s 6ms/step - loss: 0.0052
Epoch 10/100
```

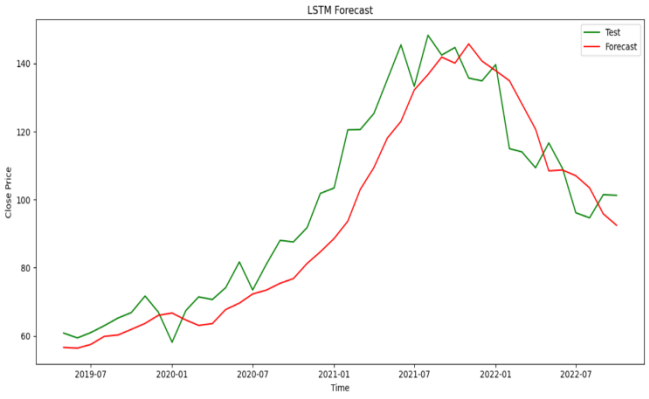

Fig 2.7

For time series forecasting, we explored creating and training an LSTM (Long Short-Term Memory) model using TensorFlow/Keras. One LSTM layer with 50 units and a ReLU activation function precedes a dense layer with one unit in the LSTM model. The Adam optimizer and mean squared error loss function are used in the model's construction. Then, with a batch size of 16, it is trained for 100 epochs using the training data (X_train and y_train). Following training, the model is applied to predict values from the test data (X_test) using the model. Inverse scaling is used to return the predictions to their original scale. The predicted values are then printed for review.

```
Epoch 1/100
4/4 [==============================] - 1s 4ms/step - loss: 0.0289
Epoch 2/100
4/4 [==============================] - 0s 6ms/step - loss: 0.0243
Epoch 3/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0203
Epoch 4/100
4/4 [==============================] - 0s 7ms/step - loss: 0.0171
Epoch 5/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0141
Epoch 6/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0118
Epoch 7/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0097
Epoch 8/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0082
Epoch 9/100
4/4 [==============================] - 0s 6ms/step - loss: 0.0070
Epoch 10/100
```

```
4/4 [==============================] - 0s 5ms/step - loss: 5.2804e-04
2/2 [==============================] - 0s 5ms/step
Forecasted Values:
[[ 56.856396]
 [ 56.38931 ]
 [ 57.39236 ]
 [ 59.842075]
 [ 60.182137]
 [ 61.695293]
 [ 63.255722]
 [ 65.3911  ]
 [ 66.16402 ]
 [ 64.48813 ]
 [ 62.71673 ]
 [ 63.11124 ]
 [ 67.071396]
 [ 68.86863 ]
 [ 71.13708 ]
 [ 72.57432 ]
```

Fig 2.8

The LSTM forecasting model was then given two evaluation metrics. The required sklearn.metrics functions mean_absolute_error and mean_squared_error are imported. It then compares the test predictions (test_predictions) with the actual test results (y_test) to get the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). The outcomes are displayed to evaluate how well the LSTM model predicted the test data.

```
Mean Squared Error (MSE): 156218491.4594597
Root Mean Squared Error (RMSE): 12498.73959483354
```

Fig 2.9

Furthermore, it states that the 'pip install prophet' command is necessary to set up the 'prophet' library, which may be used for further forecasting techniques not depicted in the provided code snippet. The 'Prophet' library was then employed to forecast time series. It creates a DataFrame from the close_ts data with the columns 'ds' for dates and 'y' for time series values. The Prophet model is then set up, trained on the data, and applied to generate forecasts for the following 365 days. Plotting the expected values reveals the anticipated trend and uncertainty ranges. The anticipated close prices are displayed throughout time in the visualization, giving users a glimpse into the time series' possible future behavior.

```
10:51:39 - cmdstanpy - INFO - Chain [1] start processing
10:51:39 - cmdstanpy - INFO - Chain [1] done processing
```

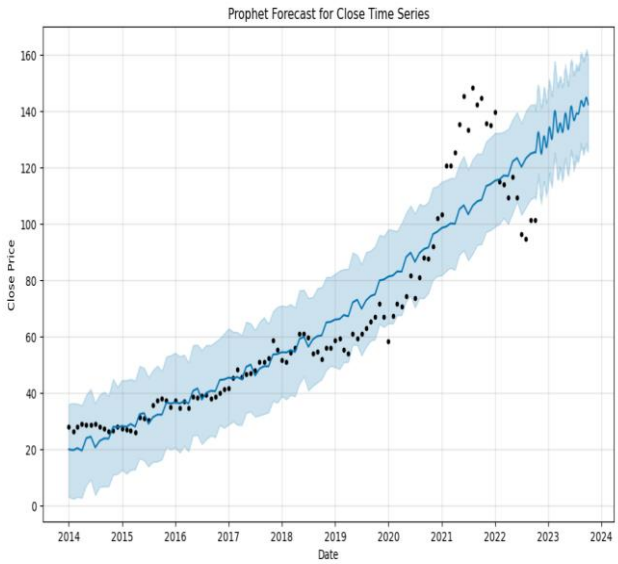

Fig 2.10

The 'Prophet' library was to be used for time series forecasting. It first prepares the data by turning the close_ts time series into a DataFrame containing columns for dates and time series values in the form of 'ds' and 'y', respectively. After that, the data is divided into training and test sets. The 'Prophet' model is developed, trained on the training set of data, then applied to forecast the test set. The Root Mean Squared Error (RMSE) and the Mean Absolute Percentage Error (MAPE) are then computed as evaluation metrics using the projected values and the actual values from the test set. According to these criteria, the 'Prophet' model's predictions were more accurate than the actual test data.

```
10:51:40 - cmdstanpy - INFO - Chain [1] start processing
10:51:41 - cmdstanpy - INFO - Chain [1] done processing

Root Mean Squared Error (RMSE): 31.272656361956184
Mean Absolute Percentage Error (MAPE): 20.913033563765204
```

Fig 2.11

Therefore, given that the Prophet technique has a low RMSE and MAPE, we may say that it is better.

CONCLUSION

This paper successfully developed a data product for stock price analysis and prediction using machine learning techniques. The data product analyzed historical stock market data of Google and applied forecasting models like Prophet and LSTM to predict future stock prices. The data preprocessing and cleaning ensured the dataset's reliability and consistency, while time series analysis identified trends and seasonality patterns. The forecasting models were evaluated using performance metrics like RMSE and MAPE to assess their accuracy.

The data product has the potential to offer valuable insights to investors and businesses in the stock market domain. By providing accurate predictions, it can enable investors to make informed decisions, manage risks, and potentially increase returns on investments. For businesses, the data product can serve as a strategic tool to identify market trends and make decisions that lead to increased profitability.

The success of the data product hinges on its ability to consistently provide accurate predictions and its user-friendliness. Positive user feedback and the achievement of low RMSE and MAPE scores indicate the data product's effectiveness in forecasting stock prices. Continuous monitoring and updating of the forecasting models will be essential to keep the data product relevant and reliable in changing market conditions.

Overall, the data product has the potential to bring significant benefits to the stock market domain, empowering investors and businesses with valuable market insights and predictions for informed decision-making.

References:

[1] Advantages of Stock Market Prediction | Benefits You Must Know. (2022, June 3). Stock Pathshala. https://www.stockpathshala.com/advantages-of-stock-market-prediction/

[2] Google Stock Data 2014-2022. (2022, December 23). Kaggle. https://www.kaggle.com/datasets/malayvyas/google-stock-data