UNIVERSITY OF
WESTMINSTER
SCHOOL OF
COMPUTER SCIENCE & ENGINEERING

Forecasting and Analysis of Stock Prices

by
PRIYAL JHUNJHUNWALA
(w1977628)

Supervised by
YONGCHAO HUANG

Submitted in partial fulfilment of the requirements of
the School of Computer Science & Engineering
of the University of Westminster
for award of the Master of Science
in
DATA SCIENCE AND ANALYTICS

SEPTEMBER 2024

DECLARATION

I, *Priyal Jhunjhunwala*, declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This has not previously been accepted as part of any other degree submission.

Signed: 

Date: 02/09/2024

FORM OF CONSENT

I, *Priyal Jhunjhunwala*, hereby consent that this Project, submitted in partial fulfilment of the requirements for the award of the MSc degree, if successful, may be made available in paper or electronic format for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed: 

Date: 02/09/2024

List of Figures

Figure 76: Volume MSE

Abbreviations

ADF: Augmented Dickey-Fuller

API: Application Programming Interface

ARIMA: AutoRegressive Integrated Moving Average

AWS: Amazon Web Services

BiLSTM: Bidirectional Long Short-Term Memory

CNN: Convolutional Neural Network

LSTM: Long Short-Term Memory

MAPE: Mean Absolute Percentage Error

MSE: Mean Squared Error

RMSE: Root Mean Squared Error

RNN: Recurrent Neural Network

SVM: Support Vector Machine

SVR: Support Vector Regression

VAR: Vector Autoregression

XGBoost: Extreme Gradient Boosting

ABSTRACT

This study investigates the application of advanced machine learning techniques to forecast and analyse stock prices, focusing on methods such as Multilayered Bidirectional Long Short-Term Memory (BiLSTM), Support Vector Machine (SVM), XGBoost, and Vector Autoregression (VAR). Stock price forecasting is a highly complex task due to the multitude of influencing factors such as market volatility, economic indicators, and temporal dependencies. This project aims to develop models capable of predicting stock prices for upcoming periods (for example, days or weeks), leveraging historical data and machine learning algorithms.

The methodology consists of multiple stages, beginning with data preprocessing. Historical stock price data is scaled and transformed, with a particular emphasis on time-series analysis to capture temporal dependencies. Techniques such as sequence splitting and moving averages are employed to prepare the dataset. Feature extraction methods, including the creation of relevant time-series data, improve the predictive accuracies of the models like BiLSTM, which is adept at handling temporal patterns in data. XGBoost, on the hand, is employed for its strong capability in handling structured datasets and providing interpretable results. Meanwhile, SVM is employed to capture non-linear relationships in stock price movements.

The implementation focuses on several models trained and tested on real-world stock data assessing performance using a key evaluation metric, Mean Squared Error (MSE). Results of this study demonstrate the efficacy of each model in predicting stock prices, with each method offering strengths in their specialized different contexts.

Visualizations of predicted versus actual stock prices are provided for each model and feature, alongside detailed performance comparisons. The analysis reveals both the potential and the challenges of using machine learning in stock price forecasting, emphasizing the importance of feature selection, hyperparameter tuning, and the handling of stationarity of time-series data. Finally, limitations such as overfitting and data dependency are discussed, with future directions highlighted to improve model robustness and scalability.

This project thus serves as a comprehensive analysis of various machine learning approaches to stock price forecasting, offering insights into their practical application, accuracy, and future improvements.

Contents

1. INTRODUCTION

Stocks, also known as shares or equities, are representative of one's ownership in a company and serve as a fundamental component of modern financial markets. When individuals or institutions buy stocks, they are essentially purchasing a small portion of the company, entitling them to a share of its profits, often in the form of dividends, and the potential capital gains. The importance of stocks lies in their dual role: they provide the companies with a means of raising capital for expansion and innovation, while offering investors opportunities for wealth creation through price appreciation and dividends. Thus, stock markets play a pivotal role in the economy by facilitating investment, fostering economic growth, and enabling efficient capital allocation.

However, the stock market is an inherently complex, dynamic and volatile environment where the price fluctuates on the basis of a myriad of factors, including but not limited to economic indicators, global events, company performance, and investor sentiments. With the growth and development in the financial and investment industries, and the consequential growth in knowledge regarding investments and wealth creation, there has been a substantial rise in the investors in the stock market. Investors and analysts have long been drawn to the challenge of forecasting these price movements, as even slight improvements in prediction accuracy can result in significant financial gains. Traditional methods of stock market analysis primarily involve fundamental and technical analysis. Fundamental analysis focuses on evaluating a company's intrinsic value based on its financial statements, earnings reports, and overall economic conditions. In contrast, technical analysis vastly relies on historical prices and trading volume data to identify patterns and trends that may be indicative of future price movements. While these approaches have been widely used for decades, they often struggle to account for the growing complexity and volume of stock market data, especially in the face of unpredictable market events.

In recent years, advancements in machine learning have offered a new paradigm for stock price prediction. Machine learning models have the capability to analyse vast amounts of data, identify non-linear relationships, and uncover hidden patterns that are often missed by traditional methods. This project aims to explore the application of several machine learning techniques in order to forecast and analyse stock prices. We aim to achieve this by leveraging historical market data and advanced computational models to develop predictive algorithms that can provide with actionable insights into the future stock movements. The analysis encompasses a range of machine learning methods such as Long Short-Term Memory (LSTM), Support Vector Regression (SVR), Time Series Forecasting with Extreme Gradient Boosting (XGBoost) and Vector Autoregression (VAR) model, to forecast Tesla's stock prices. The rationale behind choosing machine learning methods lies in their ability to process large datasets, learn from historical data, and adapt to changing market conditions. Unlike traditional approaches, machine learning methods can handle the complexity and volatility of stock price

movements more effectively, providing more accurate and timely predictions. By comparing the performance of these advanced algorithms, we seek to identify which model delivers the best results.

The key findings of this project include insights into the strengths and weaknesses of each model in predicting stock prices, the degree to which they can generalize to unseen data, and their ability to capture short-term versus long-term trends. For instance, BiLSTM, with its ability to capture both forward and backward dependencies in time series data, is expected to outperform other models in capturing complex temporal patterns. Meanwhile, SVR could offer better results in scenarios with less volatility but may struggle with large datasets or high fluctuations. XGBoost, while powerful for structured data, may face challenges with sequential data, whereas VAR, being a traditional statistical model, might falter when faced with the non-linearity and high volatility of stock prices.

In conclusion, this project aims to advance our understanding of machine learning's role in financial forecasting by comparing multiple approaches to stock price prediction. Through comprehensive analysis and evaluation, we hope to contribute valuable insights into the future of stock market forecasting and guide investors, analysts, and researchers in making more informed decisions.

2. BACKGROUND

Stock market prediction has been the central topic of interest in the realm of finance and economics for decades. The traditional approaches to forecasting stock prices have previously relied on a fundamental analysis, which examines economic and financial factors, and technical analysis, which focuses on historical price patterns and market indicators. While these methods have provided valuable insights, they have also often fallen short in capturing the complex, nonlinear relationships that can influence stock prices. Such limitations have led to the exploration of more advanced computational techniques.

The recent advancements in machine learning have opened up many new avenues for financial analysis. Machine learning, a subset of artificial intelligence, involves training the algorithms in order to help them recognize patterns and make decisions based on the data. Unlike many conventional statistical methods, machine learning methods can help process large volumes of data, recognize intricate patterns, and adapt to new information. This capability makes it particularly suitable for stock market forecasting, where vast amounts of historical data are required to be processed along with the consideration of various intangible and societal factors that influence these fluctuations.

In context to the financial markets, there is vast variety of machine learning methods which can be used for different purposes and desired outcomes, these methods come from the three major classifications of machine learning models: supervised, unsupervised and reinforcement learning approaches. Supervised learning models, such as linear regression, decision trees, and support vector machines, are trained on historical data with known outcomes to predict future stock prices. Unsupervised learning techniques, including clustering and dimensionality reduction, help in identifying underlying structures in data without any prior knowledge of the outcomes. Lastly, reinforcement learning, which is a more recent development, involves models that learn optimal strategies through interactions with the environment making them suitable for algorithmic trading and portfolio management.

The application of machine learning methods in stock price prediction have shown considerable amount of promise in various aspects. Neural networks, for instance, particularly deep learning models, have demonstrated superior performance in capturing complex, nonlinear dependencies in stock market data. Additionally, ensemble methods, that allow us to combine the predictions of multiple models, have proven effective in enhancing forecast accuracy and robustness.

Despite potential advantages, the use of machine learning in stock market analysis also presents challenges. The financial markets are influenced by a melange of unpredictable factors ranging from geopolitical events to regulatory changes to market sentiments. This leads to high volatility and uncertainty. Moreover, the risk of overfitting, where the models perform extremely well on training data but fail to generalize and provide a form of actionable outcome on unseen data, is a significant concern. Consequently, rigorous model validation and testing are crucial in order to ensure reliable predictions.

This project seeks to delve into the application of various machine learning methods to the task of stock price forecasting. Through comparing the performances of different models on the different components

of the stock price data, and identifying the key features that influence stock movements, we aim to contribute to the understanding of how machine learning can be effectively employed in financial market analysis. This work builds on a growing body of literature that explores the intersection of finance and data science, highlighting both the opportunities and the challenges of this innovative approach.

3. LITERATURE REVIEW

Forecasting stock prices remains one of the most challenging endeavours in financial analytics due to the complex and volatile nature of the financial markets. Over the past few decades, the advancements in machine learning have significantly enhanced the methodologies and techniques used for predicting stock prices, moving beyond traditional statistical approaches to embraces the more sophisticated computational techniques. This literature review examines the recent studies on various machine learning methods, focusing majorly on their application to financial time series forecasting and comparing their effectiveness.

The article, “A comparative analysis of Forecasting Financial Time Series Using ARIMA, LSTM, and BiLSTM”, explores the application, advantages and limitations of traditional methods like ARIMA in contrast to the more recently developed methods like Long Short-Term Memory (LSTM) and Bidirectional LSTM (BiLSTM). Historically, statistical models such as the AutoRegressive Integrated Moving Average (ARIMA) have been widely used for financial time series forecasting. ARIMA models are particularly effective for linear time series data and have been a staple in economic forecasting for decades. However, their limitations become apparent when dealing with the nonlinearity and complexity of stock price movements. Traditional models like ARIMA often struggle to capture intricate patterns in financial data, which can be influenced by numerous factors including market sentiments, economic indicators, and geopolitical events (Siami-Namini et al., 2019).

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), has emerged as a powerful tool for forecasting stock prices due to their ability to capture long-term dependencies in time series data. LSTMs are designed to address the vanishing gradient problem encountered in standard RNNs, allowing them to retain information over long sequences. This characteristic makes them particularly suitable for financial time series where the past data can significantly influence the future price movements (Siami-Namini et al., 2019).

Bidirectional LSTM (BiLSTM), which is an improvement made on the unidirectional LSTM, processes data in both forward and backward directions, further enhancing the forecasting accuracy by incorporating information from both past and future contexts. This bidirectional approach helps capture the full range of dependencies in financial time series, further improving the model’s ability to predict future price trends based on a more comprehensive understanding of historical patterns (Siami-Namini et al., 2019).

In addition to LSTM and BiLSTM models, advanced studies have explored the use of multi-layered LSTM networks to improve prediction performance. These models leverage hierarchical structures to learn complex patterns and interaction within the data, which are critical for accurate stock price forecasting (Zaheer et al., 2023)

This exploration of the advantage of BiLSTM over unidirectional LSTM, and the use of these methods in a layered form aids in the use and exploration of BILSTM with multiple layers for the purposes of this project, and in order to add to the existing literature.

The article, “A Comparison between Regression, Artificial Neural Networks and Support Vector Machines for Predicting Stock Market Index” talks about how Support Vector Machines (SVM) offer a robust alternative to traditional statistical methods by providing powerful classification and regression capabilities. SVMs are particularly effective in high-dimensional spaces and can model complex nonlinear relationships through the use of kernel functions. In the context of stock price forecasting, SVMs can handle the intricate patterns of financial data and deliver accurate predictions (F. et al., 2015).

The effectiveness of SVMs in financial forecasting is explored in the article, “Financial time series forecasting using support vector machines”. This study shows that the effectiveness of Support Vector Machines (SVMs) is attributed to their ability to manage large datasets and their robustness against overfitting. However, SVMs require careful tuning of hyperparameters, such as the choice of the kernel and the regularization parameter, to achieve optimal performance. Despite these challenges, SVMs have been shown to outperform the traditional models in a variety of forecasting scenarios (Kim, 2003).

Since the aforementioned literature on Support Vector Machines (SVMs) strongly supports their effectiveness and power due to their robustness, we employ the use of Support Vector Regression (SVR), a form of SVMs for the purpose of this study to further explore their advantage for our dataset and add to the existing literature on SVMs.

The article, “Stock-Price Forecasting Based on XGBoost and LSTM”, delves into the use of Extreme Gradient Boosting (XGBoost) and LSTM for the purpose of stock price forecasting. This integration of advanced machine learning and deep learning techniques significantly improved the stock price forecasting accuracy. This study applies XGBoost as a feature selection mechanism to handle high-dimensional financial data, followed by Long Short-Term Memory (LSTM) model for time-series forecasting. The findings here demonstrate that integration these two methods not only improves stock price prediction accuracy but also effectively reduces redundant data, enhancing the overall model efficiency and robustness (Vuong et al., 2022).

Such an approach is aligned with the growing recognition of hybrid models in stock price forecasting. Similar to the previously discussed works, this study highlights the effectiveness of preprocessing through machine learning before applying a deep learning architecture. The selected features focus on extracting the most relevant economic indicators, reducing noise, and optimizing the LSTM’s ability to model complex temporal dependencies (Vuong et al., 2022).

Vector AutoRegressive (VAR) models are another classical approach for multivariate time series forecasting. VAR models extend the univariate ARIMA approach to handle multiple time series simultaneously, capturing the interdependencies between different variables. This capability is particularly useful for financial forecasting, where multiple factors can influence stock prices. VAR model have been applied in the studies, “Forecasting time series using Vector Autoregressive Model” and

“Forecasting with vector autoregressive (VAR) models subject to business cycle restrictions”, to analyse relationships between stock prices and economic indicators, providing valuable insights into the broader economic context (Abdullah, 2022; Simkins, 1995).

Vector Autoregressive models are well suited for capturing dynamic interactions among variables, making them useful for understanding how various economic factors jointly influence stock prices. However, they can become complex and computationally intensive when dealing with large datasets or a high number of variables.

The recent advancements in machine learning have led to a significant amount of improvement in forecasting accuracy through optimization techniques and hybrid models. For instance, optimizing LSTM networks involves the adjusting of various hyperparameters such as the number of layers, units per layer, and learning rates. Such optimizations are crucial for the enhancement of model performance and ensuring that the network generalizes well to the unseen data (Rokhsatyazdi et al., 2020).

The integration of additional data features, such as the intraday high-low prices, has also been shown to significantly improve forecasting performance. Incorporating a high-frequency data enables models to capture the finer nuances of market behaviour, which is essential for predicting asset volatility and managing the financial risk effectively (Bai et al., 2023).

Moreover, hybrid models that combine LSTM with Convolutional Neural Networks (CNNs) or other techniques have shown promise in enhancing predictive accuracy. These models leverage the strengths of multiple approaches to capture both spatial and temporal features of financial data, leading to a more robust and reliable forecast (Selvin et al., 2017).

Comparative studies have highlighted the strengths and limitations of different forecasting methods. LSTM networks have been found particularly effective in capturing temporal dependencies and handling complex, nonlinear relationships in financial data. SVMs, on the other hand, excel in managing high-dimensional data and providing robust predictions, although they require careful tuning. VAR models are valuable for understanding the interactions between multiple time series variables but may become complex with larger datasets.

Overall, the integration of machine learning techniques into financial forecasting represents a significant advancement over traditional approaches. By leveraging the strengths of the various models and optimizing their performances, researchers and practitioners can achieve more accurate and reliable predictions. The ongoing development of new methodologies and hybrid approaches continue to push the boundaries of what is possible in financial time series forecasting, providing with promising avenues for future research and applications.

4. PROBLEM STATEMENT

Stock price forecasting is a critical challenge in the financial markets due to their highly volatile and dynamic nature. The price movements in the markets are influenced by complex factors including economic, political, and societal conditions. Traditional statistical models such as AutoRegressive Integrated Moving Average (ARIMA) often struggle to capture such nonlinearities and stochastic dependencies. In order to improve prediction accuracy, this project delves into the use of more advanced machine learning methods, including Multilayered Bidirectional Long Short-Term Memory (BiLSTM), Support Vector Regression (SVR), Time Series with XGBoost, and Vector Autoregression (VAR) models.

Each of these methods offers unique sets of strengths. BiLSTM is a method capable of handling temporal dependencies in financial data, capturing long-term trends in both directions, forward and backwards, of the time series. SVR, which is an extension of Support Vector Machine (SVM), can be used to solve regression problems similar to the one we have at hand in this project. It optimizes a function by finding a tube that approximates a continuous-valued function while minimizing the prediction error (Comito and Pizzuti, 2022). XGBoost ensures that only the most relevant predictors are used in forecasting through feature selection, while Vector Autoregressive (VAR) models, in contrast, analyse the dynamic relationships among multiple financial variables, making them well-suited for multivariate time-series forecasting.

Despite such advances, predicting stock prices accurately remains a challenging endeavour due to the unpredictability and variability of financial markets. Therefore, this project aims to evaluate and compare the effectiveness of these models in predicting stock prices. The objective is to identify the method that provides with the most accurate and reliable forecasts, ultimately contributing to improved decision-making for investors and financial analysts.

4.1 OBJECTIVES

The main objectives of this project are highlighted as follows:

- Develop advanced machine learning/deep learning models: one of the main objectives of this project is to build a robust and scalable machine learning/deep learning architecture that allows us to forecast stock prices in real-time, by adding on to the historical data, addressing the dynamic and volatile nature of financial markets.
- Real-time forecasting for short-term horizons: implement models capable of making precise short-term predictions, such as forecasting stock prices for the upcoming week, based on real-time input data, by adding the present values to the historical data.
- Model comparison and evaluation: evaluate and demonstrate the forecasting abilities of multiple machine learning methods (e.g., Multilayered BiLSTM, SVR, XGBoost, and VAR) using a dataset representing and organization’s day-to-day stock prices. This includes

the analysis of each model's strengths and weaknesses in capturing the relevant price trends.

- Performance analysis using Evaluation Metrics: analysing the performance of each of the forecasting models developed throughout the course of the project using the appropriate evaluation metrics such as Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE), ensuring a comprehensive understanding of their prediction accuracy and reliability.
- Identifying and addressing challenges: identifying the key challenges encountered during model development and implementation is one of the key underlying objectives of this project, which will help us understand problems such as data limitations, overfitting, and computational constraints, and provide solutions to mitigate these issues for improved model performance and usability.

4.2 PLAN OF WORK

The project will be executed by following a structured series of phases, each phase focusing on a critical aspect of stock price forecasting using machine learning methods. These phases are as mentioned below:

1. Literature Review and Problem Identification:
   This first phase of this project will involve studying existing research on stock price forecasting methods, particularly those using machine learning and deep learning techniques such as BiLSTM, SVM or SVR, Time Series Regression with XGBoost, and VAR. The goal is to identify gaps in current models and approaches, forming the foundation for the project.
2. Data Collection and Preprocessing:
   For this phase, historical stock price data, along with the relevant financial and economic indicators, will be collected from publicly accessible sources such as stock exchanges or financial APIs. The appropriate preprocessing steps will then be executed for handling missing values, normalizing datasets, and extracting key features, ensuring the data is clean and ready for input into algorithms for model building. Feature engineering will be key in transforming raw data into forms that the models can best utilize.
3. Model Development:
   The following models will be developed to forecast stock prices:
   - Multilayered Bidirectional Long Short-Term Memory (BiLSTM): This method will be implemented to capture complex temporal tendencies in stock price data. The model will process sequences of past stock prices and predict future price movements in both forward and backward directions.
   - Support Vector Regression (SVR): SVR will be implemented to predict nonlinear relationships between stock prices and their influencing factors.

- XGBoost for Time Series Regression: A gradient boosted decision tree algorithm will be used for time series forecasting, focusing on feature selection and improving the prediction accuracy by eliminating redundant variables.
- Vector Autoregression (VAR): A VAR model will be used to forecast stock prices based on the relationships between the multiple features in the dataset, such as the stock's high and low prices, and adjusted closing rates.

4. Model Training and Testing:
   The data will be split into training and test sets. Each model will be trained on historical stock price data, and various hyperparameters will be optimized to ensure that the models generalize well to unseen data. The testing phase will ensure that the trained models perform robustly on new data point.
5. Forecasting Implementation:
   A key goal in this project is to enable the models to predict stock prices in present, allowing forecasts for the upcoming week based on incoming data. This will involve integrating the latest data into the forecasting process.
6. Performance Evaluation
   Each model's performance will be evaluated using the appropriate evaluation metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). Comparative analysis will be conducted to assess which model(s) provides the most accurate and reliable predictions.
7. Error Analysis and Challenge Identification
   This phase will focus on identifying any limitations encountered during the model development and implementation process. Issues such as overfitting, data imbalance, and computational challenges will be addressed. Strategies to overcome these limitations will be explored to improve the final model's performance.
8. Documentation and Final Report
   All the findings, methodologies, and results will be comprehensively documented. A detailed project report will be prepared, including insights into the most effective models, performance analysis, and recommendations for future improvements in stock price forecasting.

5. METHODOLOGY

This section of the report essentially highlights the methodologies that were used during the course of this project, focusing on the steps taken for data collection, preprocessing, and implementation of machine learning and deep learning techniques to forecast stock prices. The methodology is divided into two primary sections: Data Sources and Implementation.

5.1 DATA SOURCES

The accuracy of any stock price forecast model relies largely on the quality and relevance of the data is used. Thus, for the purpose of this project, historical stock price information and the related financial features were obtained from publicly available and widely acclaimed financial platforms and APIs, such as:

- Yahoo Finance
- Nasdaq
- London Stock Exchange

The platforms provided stock price data for individual companies, including open, close, high, low, and adjusted close price. Additionally, financial indicators such as trading volumes were collected to enhance the feature set for machine learning models.

The data spans multiple years to capture both long-term trends and short-term fluctuations, ensuring that the models are able to generalize well across different market conditions.

5.2 IMPLEMENTATION

The implementation phase of the project encompasses a multitude of key steps ranging from data preprocessing to error analysis and refinement:

1. Data Description and Measurement Scale

   The first step of the implementation process is to gain a better understanding of the type of data that we have gathered and visualize each of the key features individually, and compare them to one another where appropriate. In order to gain a better understanding of the scales of the data, we use the describe() function (Code 1, Appendix 1). The outcome from the describe() function is represented in the below figure, Figure 1.

```
              Open         High          Low        Close    Adj Close  \
count  1258.000000  1258.000000  1258.000000  1258.000000  1258.000000
mean    179.459436   183.463867   175.127184   179.387723   179.387723
std     103.137166   105.316474   100.651629   102.977323   102.977323
min      12.073333    12.445333    11.799333    11.931333    11.931333
25%      67.472166    69.502165    66.452332    69.249336    69.249336
50%     203.660004   208.220001   198.510002   203.351670   203.351670
75%     251.392498   256.570000   246.292503   251.844997   251.844997
max     411.470001   414.496674   405.666656   409.970001   409.970001

             Volume
count  1.258000e+03
mean   1.328551e+08
std    8.476043e+07
min    2.940180e+07
25%    8.042828e+07
50%    1.086468e+08
75%    1.550344e+08
max    9.140820e+08
```

Figure 1: Description and Scales of Measurement of the Dataset

The next step here is to visualize the key features of the dataset. The below figures, Figure 2 to Figure 7 represent the stock movements, for the organization over the past 5 financial years, relating to their open, high, low, close, and adjusted close prices along with the trading volumes, respectively (Code 2, Appendix 1).

Figure 2: Historical Open price of Tesla

Figure 3: Historical High price of Tesla

Figure 4: Historical Low price of Tesla

Figure 5: Historical Close price of Tesla

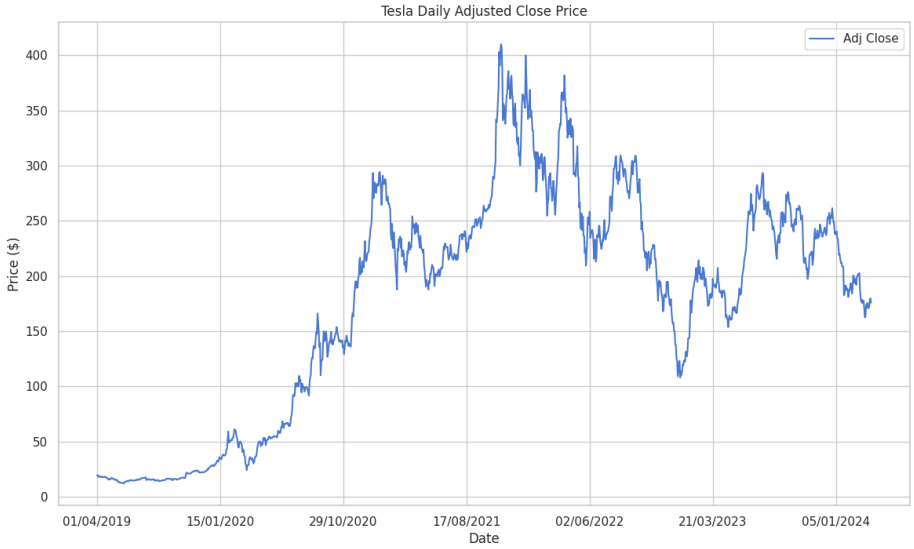

Figure 6: Historical Adjusted Close price of Tesla

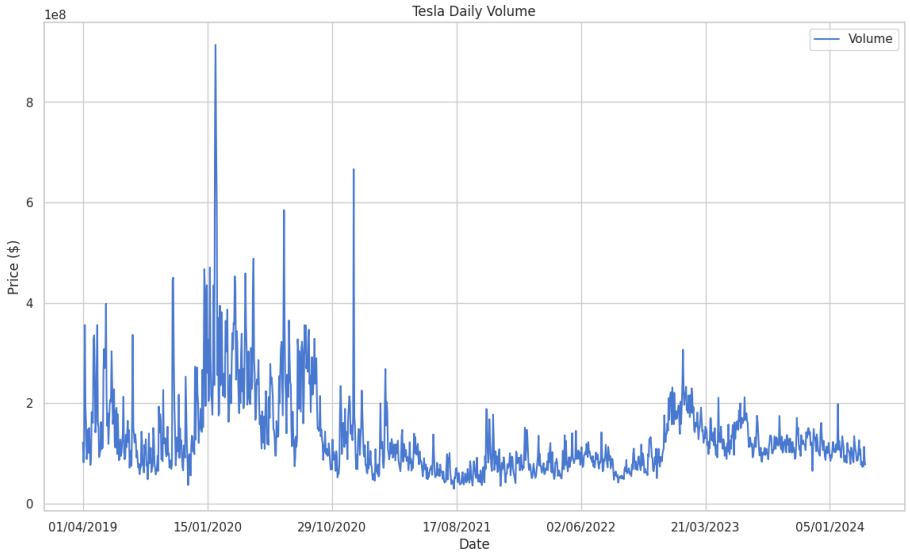

Figure 7: Historical trading Volume of Tesla

Upon such visualization of the key features, we try to observe how the values are represented when compared to one another. For such comparison, we visualize open price versus close price, and high price versus low price in figures, Figure 8 and Figure 9, respectively (Appendix 2).

Figure 8: Historical Open price vs Close price

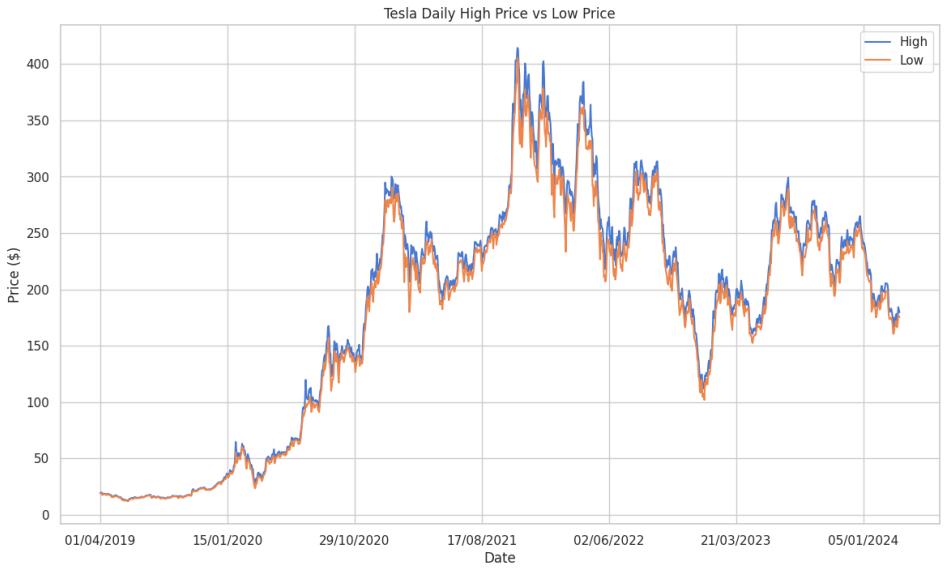

Figure 9: Historical High price vs Low price

These visualizations help us gain better understating of how the values for the features have changed over the years, and check for any irregularities and breaks in the data.

2. Data Preprocessing

   Data preprocessing is an essential step in machine learning for handling missing values, normalizing data, and ensuring that the data is clean for use in building the models for forecasting and minimizing errors. To deal with any issues regarding noisy values, outliers, and datatype issues, we use the appropriate methods to check for null values, and normalize the features using the MinMaxScaler library, storing them separately to be used with the necessary methods (Appendix 2).

3. Bidirectional Long Short-Term Memory (BiLSTM)

   The first method that we use to build a forecasting model, is the BiLSTM with multiple layers. BiLSTM is a type of recurrent neural network that addresses the long-term dependencies between time steps of time-series or sequence data. According to the aforementioned literature on this method, it has proven to be effective in predictions and forecasting. Our attempt is to gain a value interval in the forecasts for each of the features; open price, high price, low price, close price, and trading volumes. We can breakdown the building and forecasting of this model in a series of elaborate steps:

   - Step 1: We begin with this method by importing the TensorFlow library with its Keras layers and models (Code 1, Appendix 3). After importing the required libraries, we proceed with the building of the model by declaring hyperparameters and forming the training and test sets. One of the hyperparameters in the BiLSTM is the sequence length, this defines the number of past data points that will be used to predict future stock prices. We use this sequence length with a value of 120 to create sliding windows,

each of the specified length so that the function, 'splitseq', returns an array containing these sequences (Code 2, Appendix 3). The function, 'train_test_sets', generates sequences using 'splitseq'. It then divides the scaled open price data into training and testing sets based on the fraction defined by 'train_frc', which specifies that the training data should be 80% (Code 2, Appendix 3).

- Step 2: After the split, LSTM model is defined using Keras. It starts with an LSTM layer that accepts the input data with a window size of 'seq_length – 1'. This is followed by a Dropout layer, to prevent overfitting), two Bidirectional LSTM layers, which process data in both forward and backward directions, enhancing temporal dependency learning, and an Activation layer with linear activation function being used. The output of the model is a single unit representing the next predicted stock price (Code 3, Appendix 3).
- Step 3: The model is compiled using the Mean Squared Error (MSE) loss function and the Adam optimizer, which efficiently adjusts the learning rate during training. The model is trained through 30 epochs with a batch size of 16, using 20% of the data as a validation set (Code 4, Appendix 3).
- Step 4: After the training, the prepared model predicts open prices using the test set ('X_test'). The predicted and actual values are then transformed back to their original scale using the inverse scaling method for comparison (Code 5, Appendix 3).
- Step 5: The final step of the method involves visualizing the open price predictions against the actual values and the plot for changes in the MSE over 30 epochs, helping us assess the model's performance and convergence (Code 6, Appendix 3).

The aforementioned steps, Step 1 to 5, were then followed in the same manner for the forecasting of the other features in the dataset, close price, high price, low price and volume, using their respective scaled values.

4. Support Vector Regression (SVR)

The next method that we use, is the Support Vector Regression (SVR) which is an extension to Support Vector Machine (SVM). It uses the same principles as SVM but focuses on predicting continuous outputs rather than the classification of the data point. According to the aforementioned literature regarding the use of SVMs for financial forecasting, it has proven to be effective due to their ability to manage large datasets and their robustness towards unseen data. Through this model we attempt to forecast the prices for each of the features of the stock price, open, close, high and low prices and the trading volume. The development of this model and the forecasting can be broken down into a range of elaborate steps:

- Step 1: The first step is to import the SVR from 'sklearn.svm' (Code 1, Appendix 4). Then we initialize with a radial basis function (RBF) kernel. The hyperparameters, gamma, C, and epsilon are initialized with the values 0.5, 10, and 0.05 respectively, to control the model's flexibility and error tolerance (Code 2, Appendix 4).

- Step 2: After initializing the model, we split the scaled open price into training and testing sets manually (Code 3, Appendix 4). The 'create_dataset' function generates sequences of data into a look-back window of 5 timesteps. This is done to prepare the data to be input into the SVR model, which requires fixed-length input sequences (Code 4, Appendix 4).
- Step 3: The model is then trained on a reshaped 'X_train' and the corresponding 'y_train' values using the fitting function (Code 5, Appendix 4). After training, the model is used to perform predictions on both the training and the testing data to evaluate and check for overfitting and errors margins (Code 6, Appendix 4).
- Step 4: The predictions from the model and the actual values are scaled back to their original price range using the inverse scaling method (Code 6, Appendix 4).
- Step 5: The Mean Squared Error (MSE) is calculated for both training and test sets; this provides an assessment of the model's accuracy (Code 7, Appendix 4).
- Step 6: The predicted values for both the training and test sets are plotted respectively against the actual values for visual comparison. This allows us to visually determine the model's performance on unseen test data (Code 8, Appendix 4).

The aforementioned steps, Step 1 to 6, were then followed in the same manner for the forecasting of the other features in the dataset, close price, high price, low price and volume, using their respective scaled values.

5. Time Series Regression with Extreme Gradient Boosting (XGBoost)

   Time Series Regression, as the name suggests, is a regression method that is used when the input data is in the form of a time series, i.e., the data points are recorded over a period of time and have specific dates, months or years. This is a statistical method used for predicting future values based on historical data and identify patterns in a series of variables over time. We combine the time series with extreme gradient boosting. By using this model, we aim to forecast the future values of the relevant features of stock price. The implementation and execution of this model can be divided into a series of steps:

   - Step 1: The first step is to install the PyCaret library to simplify the machine learning workflow (Code 1, Appendix 5). A 5-year moving average (MA5) for the open prices is created using the rolling mean method, representing an average of the past two data points. This helps smooth the data to identify trends. The moving average and the open price data is then plotted using a grid line plot for better visualization (Code 2, Appendix 5).
   - Step 2: The code adds a column 'Series', which is a sequence of number to index the data. Other feature columns that don not directly impact the open price are dropped to retain only the essential data for open price prediction (Code 3, Appendix 5). The 'Date'

column is converted to an integer in ‘ddmmyyyy’ format to simplify numerical processing for machine learning (Code 4, Appendix 5).

- Step 3: The dataset is manually split into training and testing sets with the first 900 rows as the training set. This ensures that the time-series nature of the data is respected and the test set will contain the future data points (Code 5, Appendix 5).
- Step 4: The PyCaret regression module is used to initialize the setup for the dataset. The setup function configures the environment for time series forecasting by specifying parameters like the target feature, and the type of algorithm being used, time series, which ensures that the cross validation respects the temporal ordering of the data (Code 6, Appendix 5). The XGBoost model is then created using the ‘create_model’ method, which is a popular method for time series prediction (Code 7, Appendix 5).
- Step 5: The model created from the above steps is used to generate predictions on the training and test sets using the ‘predict_model’ method. Additionally, predictions for the entire dataset are generated. A new feature, ‘Date’, is added to the predicted dataset for the purpose of visualization. A line plot is then generated to show actual vs. predicted open prices over time, with a grey grid that separates the test period from the training period (Code 8, Appendix 5).
- Step 6: The best model is then finalized and predictions for future open prices are made by generating a new data frame with date ranging from the years 2024 to 2028. The ‘Series’ and ‘MA5’ columns are also added to the future dataset, and the final model is used to predict the future prices (Code 9, Appendix 5).
- Step 7: The past and future predictions are concatenated and the dataset is indexed by the new date range. The final plot is then generated, showing both historical actual open prices and predicted future values. This final visualization allows for and intuitive understanding of how well the model can forecast stock prices into the future (Code 10, Appendix 5).

The aforementioned steps, Step 1 to 7, are then followed in a similar manner in order to forecast the other features of the dataset, namely, ‘Close’, ‘High’, ‘Low’, and ‘Volume’.

6. Vector Autoregression (VAR)

The last model we implement is the Vector Autoregression (VAR) model. This is a statistical model that is used for multivariate time series, to capture the relationship between multiple features in the dataset over time. It is a type of stochastic process model that can be used to examine the relationships between variables, which comes in handy as the various components of stock prices are not only affected by their historical values but are also sensitive to the other components. If we want to forecast the open price of a stock, it is not only affected by its own historical prices but the other components such as high, low, close prices, and the trading

volume are also instrumental in affecting the open price. The implementation of this method can be demonstrated through a series of elaborate steps:

- Step 1: We start the construction of this model by importing the Augmented-Dickey Fuller (ADF) test to check if the time series data is stationary (no trend or seasonality) as it is an automatic key assumption in VAR models (Code 1, Appendix 6). After this we start on the setting up of the dates as index (Code 2, Appendix 6), the ADF test is run for each feature in the dataset, returning the statistics such as the ADF value, p-value, and critical values. If the p-value is less than 0.05, the series is deemed as stationary; otherwise, it is non-stationary (Code 3, Appendix 6).
- Step 2: After the dataset is checked for stationarity, we employ Holt's Exponential Smoothing to model the trend component of the stock price data, 'Open', 'High', 'Low', etc. The Holt method fits a model to each price feature, estimating the level, trend, and the residuals (the difference between the actual and modelled values). This method allows separating the components of the time series data for more accurate prediction and forecasting (Code 4, Appendix 6).
- Step 3: Using seasonal decomposition, the code separates each feature into its trend, seasonal, and residual components. This is crucial for understanding patterns in the stock data, especially is seasonality is present. This helps assess whether any patterns repeat periodically (Code 5, Appendix 6).
- Step 4: The Vector Autoregression (VAR) model is used for multivariate analysis. A key component before fitting the model is checking for multicollinearity with the Variance Inflation Factor (VIF), ensuring that predictor variables are not highly correlated (Code 6, Appendix 6). The model is then fit with the selected hyperparameters (Code 7, Appendix 6). After selecting the appropriate lag length, the Durbin-Watson statistic is computed to check for autocorrelation in residuals (Code 8, Appendix 6).
- Step 5: The fitted VAR model is used to forecast the next 10 timesteps (10 days). The 'forecast()' method predicts future values for the trend and residual components of the stock prices. These predictions are combined to obtain the final forecast for the open price. Each of these historical and predicted values for the open price residual and the open price are visualized in the form of line plots, respectively. (Code 9, Appendix 6 & Code 10, Appendix 6).
- Step 6: Finally, we compute the Mean Squared Error (MSE) between the actual and predicted open prices to evaluate the accuracy of the forecast. The lower the MSE, the better the model performance (Code 11, Appendix 6).

The aforementioned series of steps, Step 1 to 6, are similarly implemented to derive the forecast results for the rest of the features in the dataset, namely, 'High', 'Low', 'Close', and 'Volume'.

6. RESULTS AND DISCUSSION

This section of the report reflects on the performance of the several machine learning models that were applied to forecast stock prices. These models, namely, Bidirectional LTSM, SVR, XGBoost, and VAR, were evaluated based on their predictive accuracy and ability to capture complex market trends. We not only discuss the model performance and the results derived from these models, but also the limitations and challenges faced during the course of this project.

The results of the project are discussed in detail with respect to the models generating them.

BiLSTM:

This method was the first one used to build a forecasting model for the dataset. This model was designed to capture any temporal dependencies. We extracted each feature from the dataset, namely, 'Open', 'High', 'Low', 'Close', and 'Volume' to forecast their future values using historical data.

Upon executing the aforementioned steps for BiLSTM, the resultant prediction graphs and error values on 'Open' are represented in figures 10 and 11 below

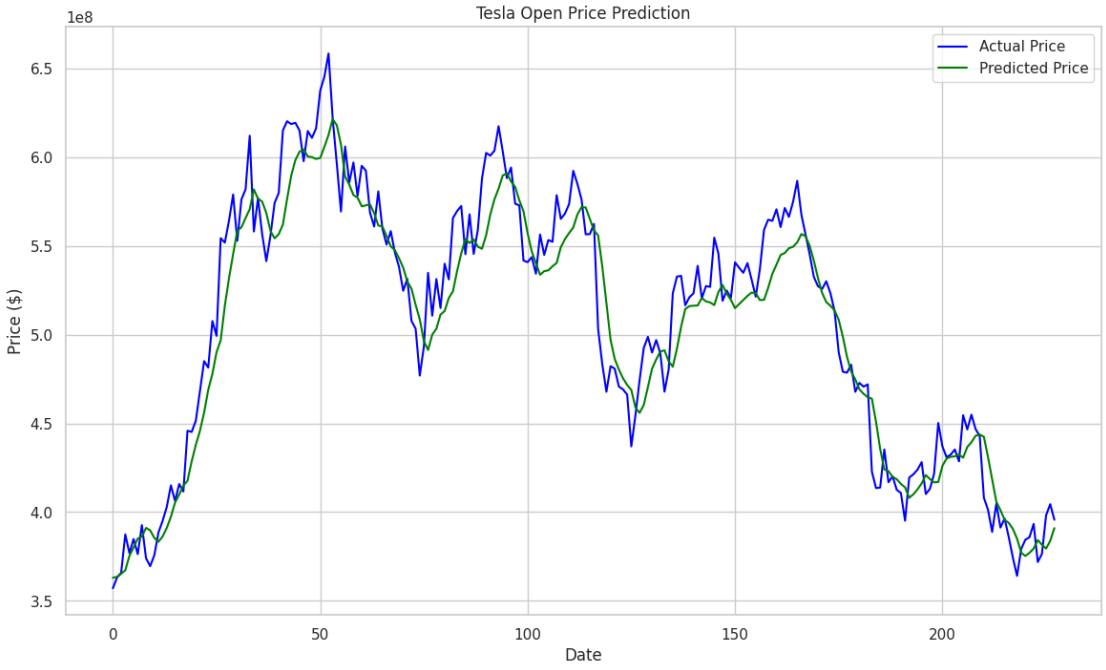

Figure 10: Open Price prediction vs actual

Figure 10, above, demonstrates the predicted values generated by the model over time in comparison to the actual values for open prices of Tesla. The close alignment between the blue line, representing actual prices, and the green line, representing predicted prices, indicates that the model performs well in capturing the general trend of Tesla's stock prices. Some minor deviations can be observed between the actual and predicted values, particularly during periods of rapid fluctuations, however, overall, the model demonstrates a strong predictive accuracy.

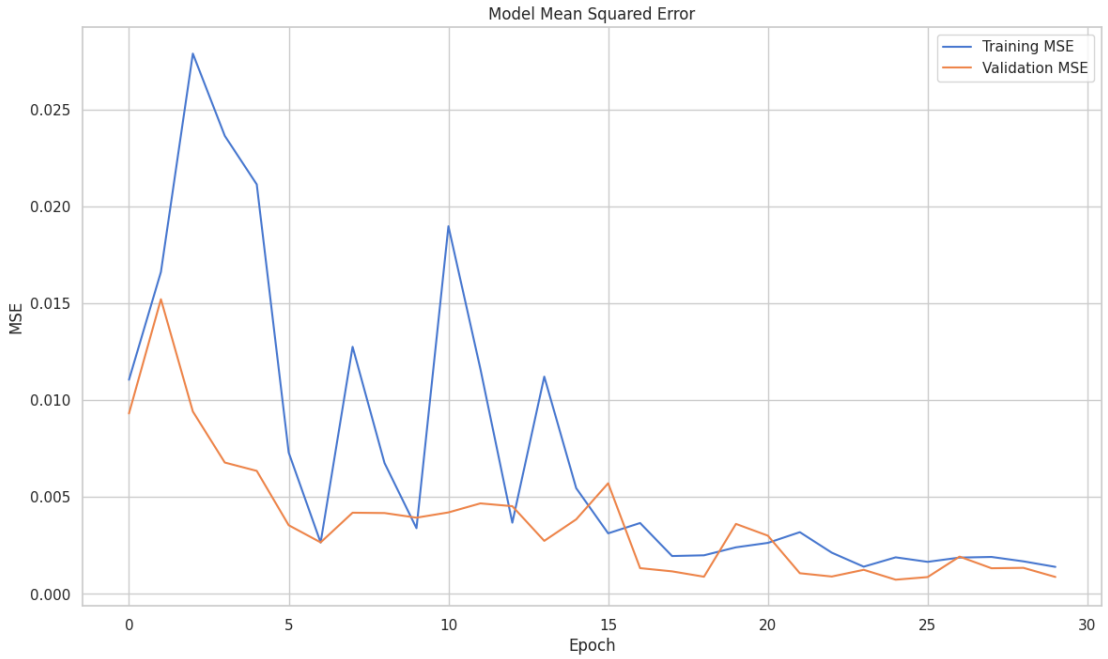

Figure 11: MSE for 'Open' through 30 epochs

The above plot, Figure 11, illustrates the Mean Squared Error (MSE) for both training and validation datasets across several epochs. Initially, both training and validation MSE values fluctuate significantly, with higher error values. However, as the number of epochs increases, the error rates stabilize and decrease due to the use of the optimization function Adam, which adjusts the learning rate adaptively based on the history of gradients calculated with the loss parameter being MSE. It is observed that by the 25th epoch, the MSE for both sets is quite low, indicating that the model's performance improves over time. Notably, the validation error closely tracks the training error, suggesting minimal overfitting and strong generalization to unseen data. Figure 12, below accurately demonstrates the final values of the MSE for both the training and validation sets which are considerably low.

```
Final Training MSE: 0.0013961843214929104
Final Validation MSE: 0.0008764723897911608
```

Figure 12: Final Training and Validation MSE on Open prices

Similarly, we observe that the model performs equally as well with the features, 'Close', 'High', and 'Low', demonstrating its capability to capture trends and non-linear relationships among the data points. This can be also be observed through the resultant figures below:

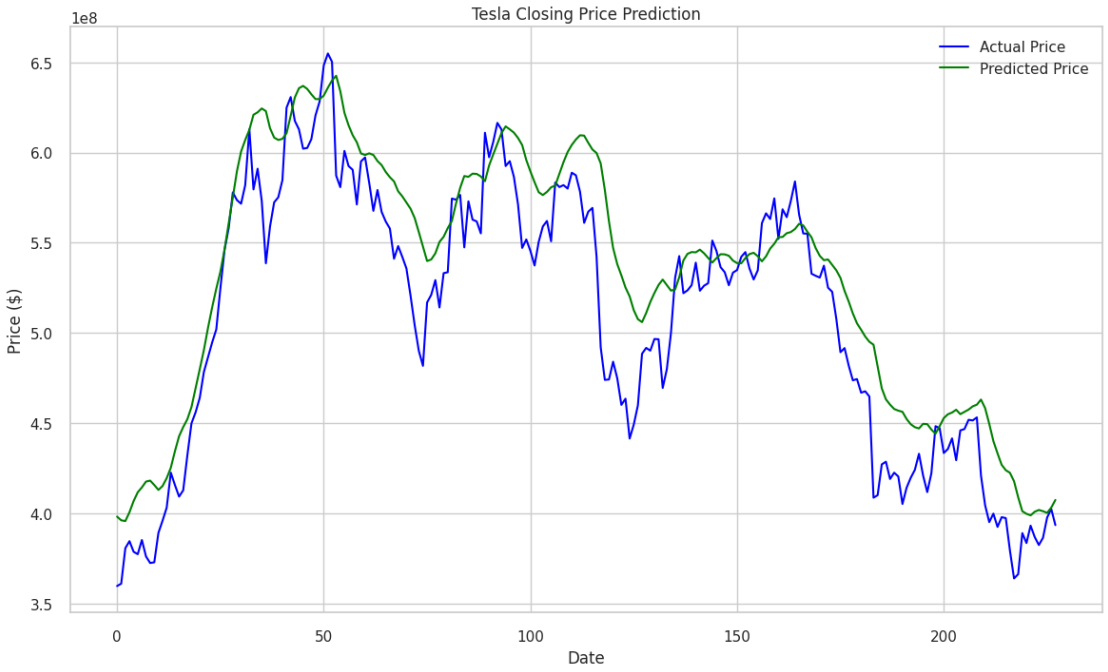

Figure 13: Close Price prediction vs actual

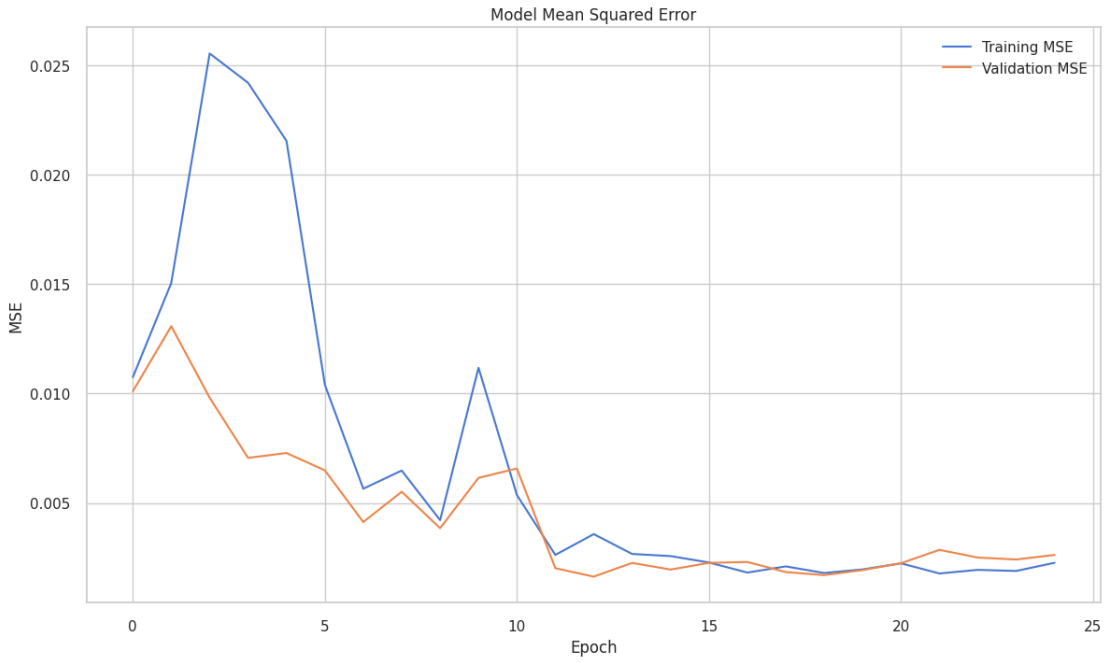

Figure 14: MSE for 'Close' through 25 epochs

```
Final Training MSE: 0.0022673816420137882
Final Validation MSE: 0.0026279871817678213
```

Figure 15: Final Training and Validation MSE on Close prices

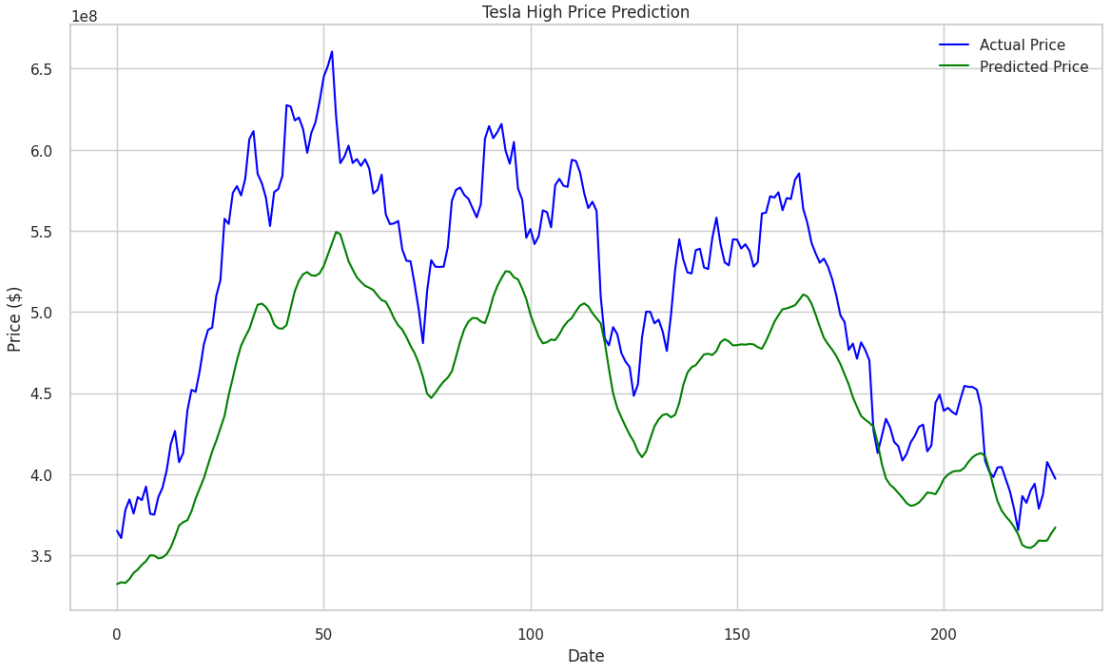

Figure 16: High Price prediction vs actual

Figure 17: MSE for 'High' through 20 epochs

```
Final Training MSE: 0.0026287452783435583
Final Validation MSE: 0.0031434556003659964
```

Figure 18: Final Training and Validation MSE on High prices

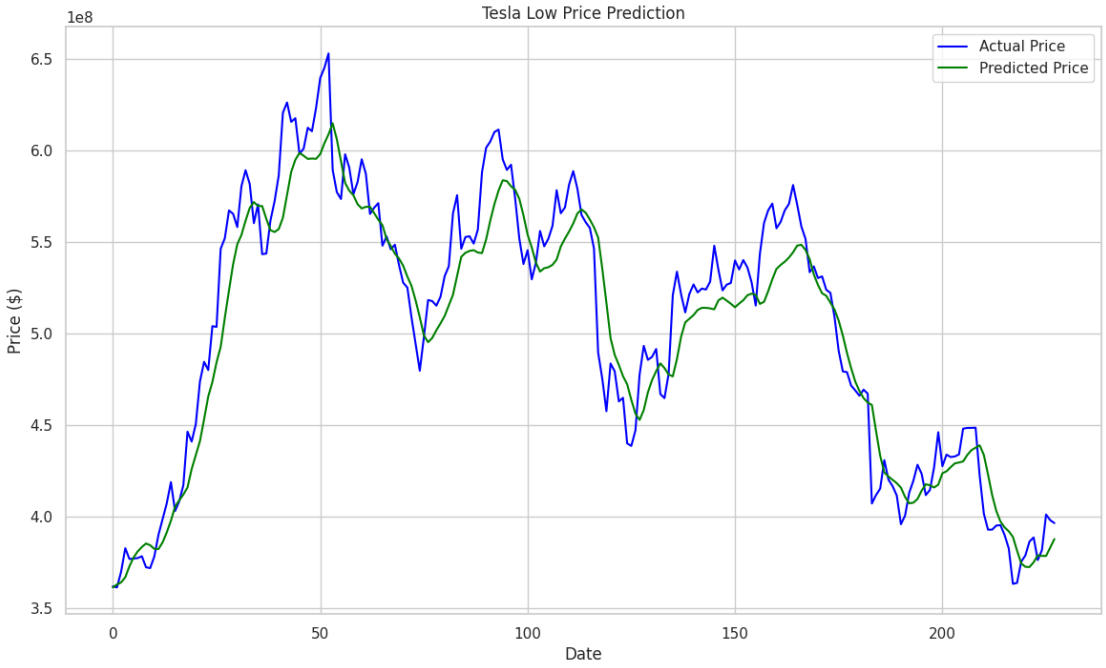

Figure 19: Low Price prediction vs actual

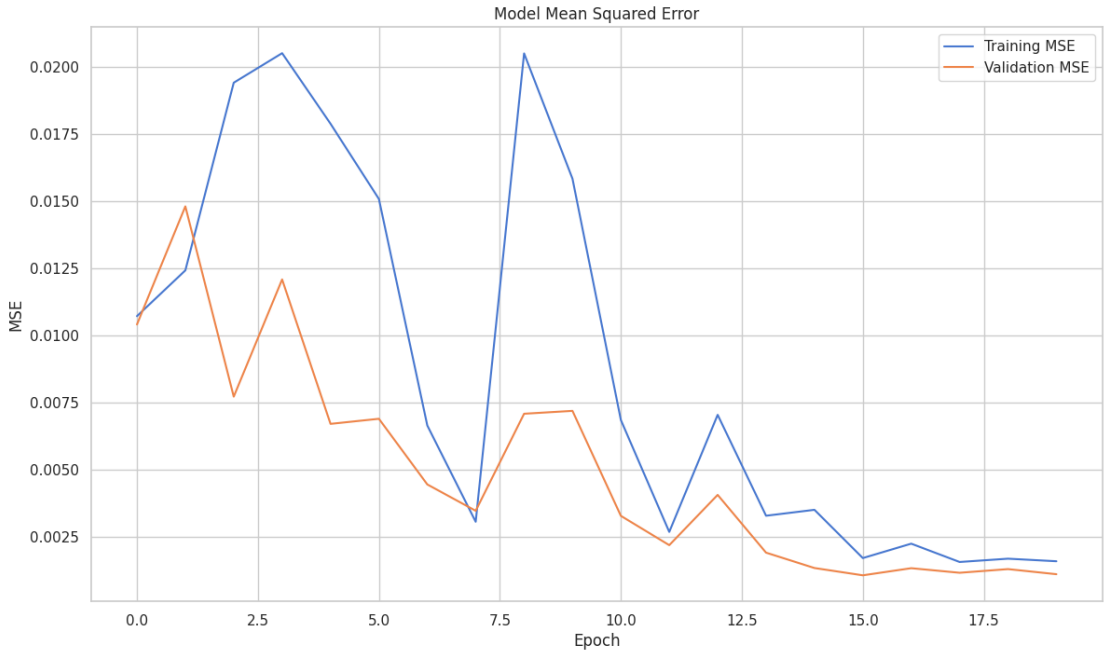

Figure 20: MSE for 'Low' through 20 epochs

```
Final Training MSE: 0.0015907887136563659
Final Validation MSE: 0.0011078407987952232
```

Figure 21: Final Training and Validation MSE on Low prices

While the model performs seemingly well with the rest of the features of the dataset, it is slow in minimizing the error in case of 'Volume' due to the high and rapid fluctuations in its values over time. The error on training and validation sets, as illustrated in Figure 23, does not fluctuate as significantly as with the other features and is more stabilized from the beginning, however, it could be beneficial to run this feature's sets through more epochs to minimize the errors further.

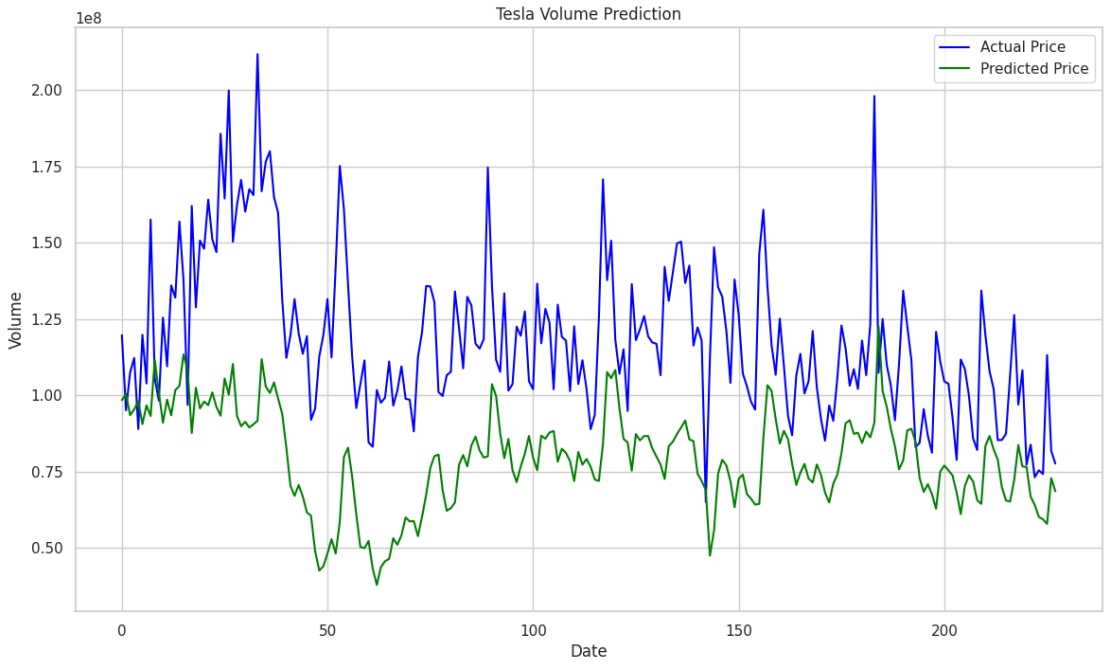

Figure 22: Volume prediction vs actual

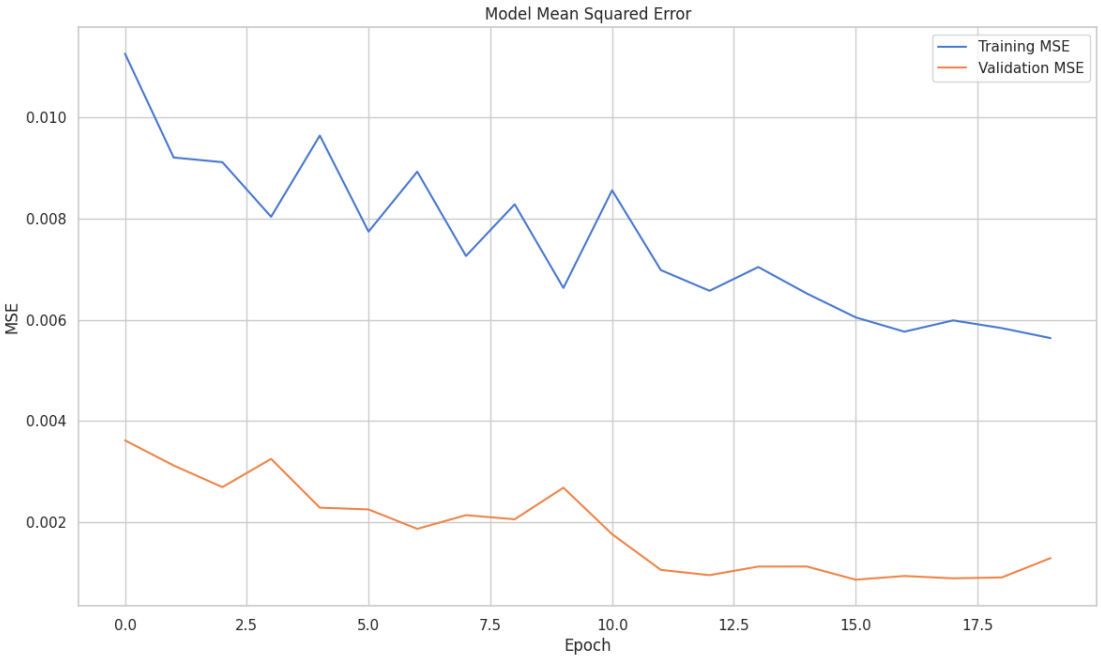

Figure 23: MSE for 'Volume' through 20 epochs

```
Final Training MSE: 0.005638534668833017
Final Validation MSE: 0.001292604603804648
```

Figure 24: Final Training and Validation MSE on Volume

In conclusion, the visualizations demonstrated that the BiLSTM model captured the overall trend, although minor deviations were present. The model has demonstrated robust performance in forecasting Tesla's stock prices, with minimal error, as evidenced by both actual-vs-predicted price comparison and the steady reduction in MSE.

SVR:

The SVR model was implemented to predict stock price movements, leveraging an RBF kernel. The training and test datasets created by splitting the stock price sequences into appropriate segments, helped the model produce reliable predictions for short-term price movements.

We extracted each feature from the dataset, namely, 'Open', 'High', 'Low', 'Close', and 'Volume' to forecast their future values using historical data.

Upon executing the aforementioned steps for SVR, the resultant prediction graphs and error values on 'Open' are represented in figures 25, 26, and 27 below.

The first chart, Figure 25, shows the predicted and original Tesla open prices on the training set. The green line represents the original prices, while the red line indicates the predicted price values. It is

observed that the SVR model tracks both long-term and short-term price fluctuations accurately, showing strong predictive capability in matching the overall stock price movement.

Figure 25: Open price prediction vs actual on Training set

The plot, Figure 26, illustrates the SVR model's predicted values on the test dataset. Once again, the model demonstrates a close approximation to actual open prices, with minimal divergence from the original values. This strong alignment between predicted and actual prices on the test data confirms the SVR model's effectiveness in generalizing beyond the training dataset.

Figure 26: Open price prediction vs actual on Testing set

The last figure, Figure 27, provides us with the MSE for training and testing datasets. Here, it is observed that the values are large, this is likely due to the scale of the data rather than poor model performance. Despite the high numerical values, the SVR model demonstrates a robust prediction ability, as visualized in the charts above.

```
Mean Squared Error on Training Data: 697012128003224.0
Mean Squared Error on Testing Data: 298971601747315.9
```

Figure 27: Open price MSE on training and test set

Similarly, we observe that the model performs equally as well with the features, 'Close', 'High', and 'Low', demonstrating its capability to capture short-term trends and provide closely accurate predictions. This can be also be observed through the respective resultant figures below:

Figure 28: Close price prediction vs actual on Training set

Figure 29: Close price prediction vs actual on Testing set

```
Mean Squared Error on Training Data: 448409406884504.1
Mean Squared Error on Testing Data: 274386063370211.84
```

Figure 30: Close price MSE on training and test set

Figure 31: High price prediction vs actual on Training set

Figure 32: High price prediction vs actual on Testing set

```
Mean Squared Error on Training Data: 614206706316846.2
Mean Squared Error on Testing Data: 223173851795973.34
```

Figure 33: High price MSE on training and test set

Figure 34: Low price prediction vs actual on Training set

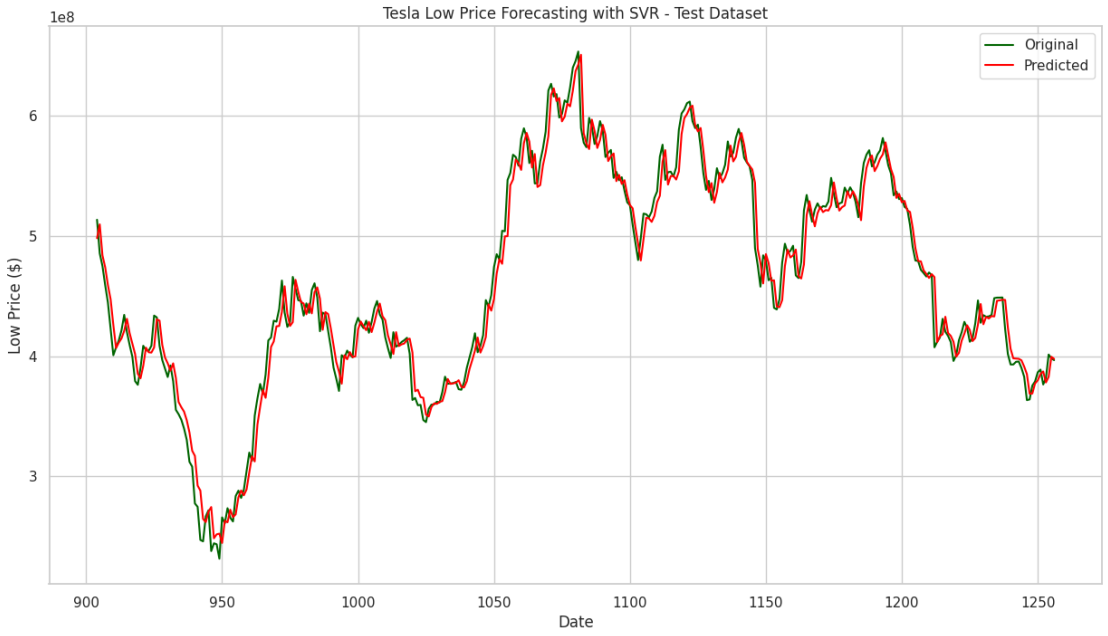

Figure 35: Low price prediction vs actual on Testing set

```
Mean Squared Error on Training Data: 634712856973209.9
Mean Squared Error on Testing Data: 224912156581134.16
```

Figure 36: Close price MSE on training and test set

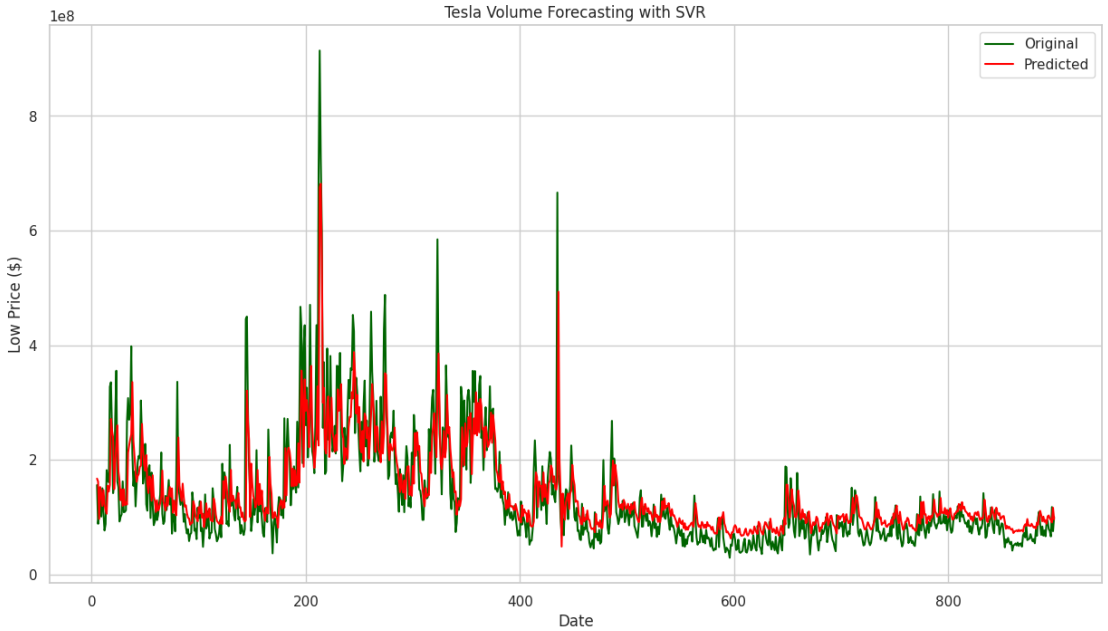

Figure 37: Volume prediction vs actual on Training set

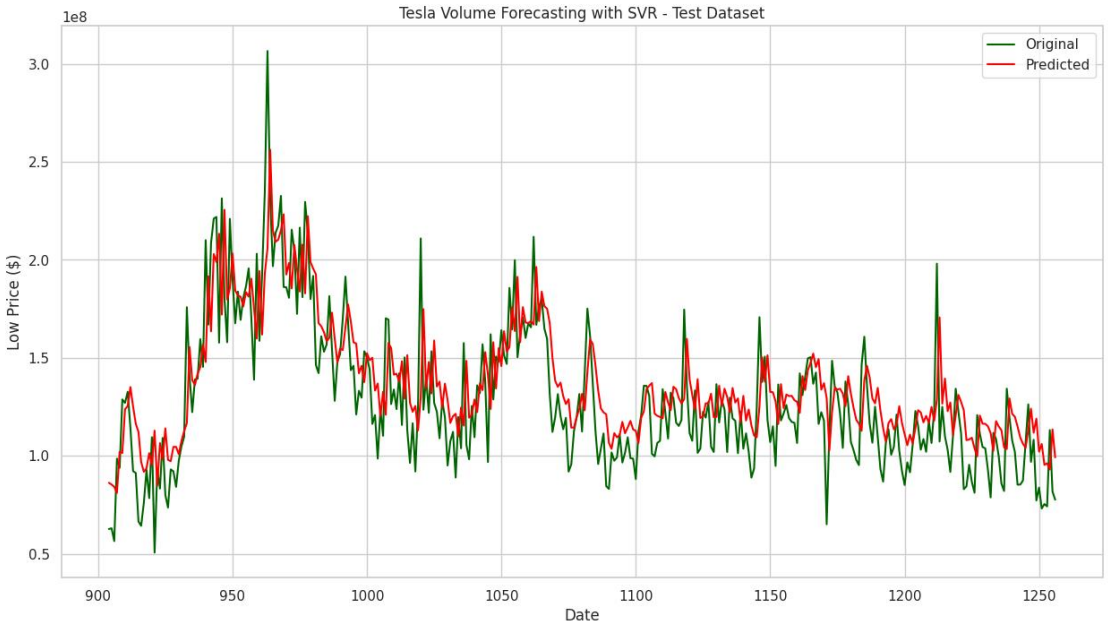

Figure 38: Volume prediction vs actual on Testing set

```
Mean Squared Error on Training Data: 3088000927304315.5
Mean Squared Error on Testing Data: 585628817017195.9
```

Figure 39: Volume MSE on training and test set

In conclusion, though the large magnitude of MSE values indicates a substantial difference in the predicted and actual stock prices, it is possibly due to the data scale or the inherent volatility in stock prices, and despite the large numbers, the visual accuracy in predictions suggests the model is effective in capturing trends. Fine-tuning and scaling adjustments might help reduce the MSE.

Time Series with XGBoost

The Time Series regression model is built with the purpose of building a model that is adept in capturing trends, recognizing patterns and providing forecasts on the basis of historical data. This model is implemented with the combination of Extreme Gradient Boosting (XGBoost). We perform feature extraction on the dataset to forecast their future values using their historical data.

Upon executing the aforementioned steps regarding this model, each feature produces 3 charts. The chart illustrated in Figure 40, demonstrates the original values of ‘Open’ plotted against the moving averages calculated for this feature using rolling mean over 2 data points, and Figure 41 demonstrates the values that fall under the test set by highlighting them in grey rectangle for better visual understanding.

Figure 40: Open price vs Moving Averages

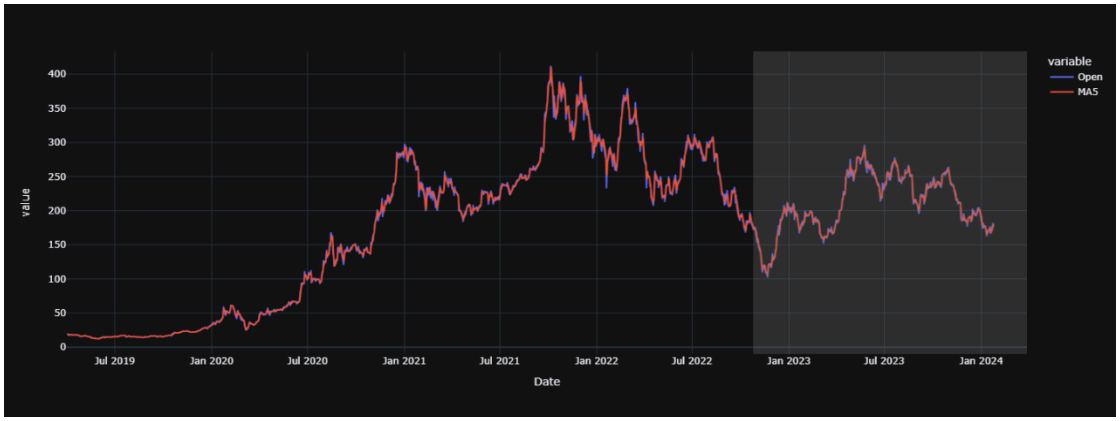

Figure 41: Visual segregation of training and test sets

It is observed from the figure below, Figure 42, that the model produces a relatively low MSE. However, when the historical Open price values and the predicted values are visualized together on a chart, as represented by Figure 43, we can observe that the predicted values appear to be following a pattern identical to the pattern of historical values with a few minor fluctuations. This leads us to believe that either the model has been overfit or the dataset has not been prepared in an ideal manner to make it suitable for this kind of model.

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Extreme Gradient Boosting	3.0887	66.4034	8.1488	0.9938	0.0523	0.0199

Figure 42: Error metrics on XGBoost for Open price

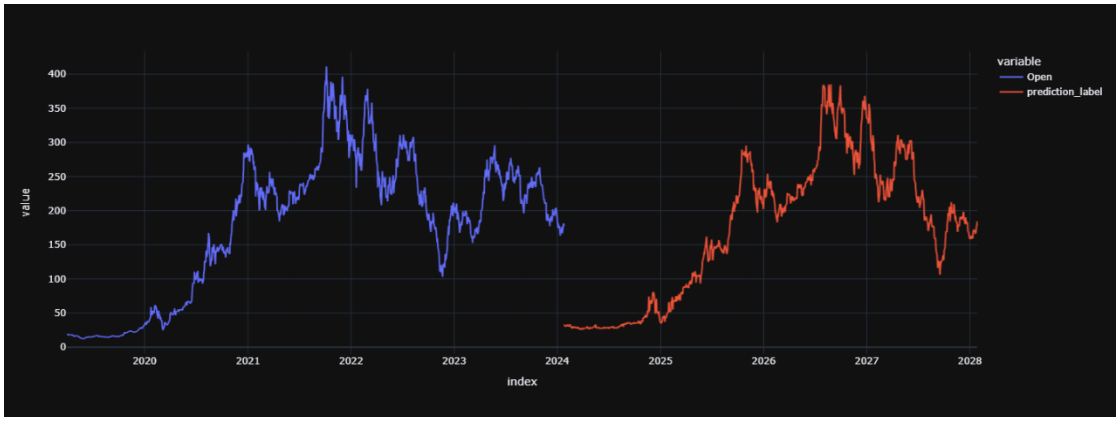

Figure 43: Actual and predicted Open prices

A similar case of the above highlighted observations is seen for the remaining features, namely, 'Close', 'High', and 'Low', where the error values for the metrics are relatively low however, the visual representation demonstrates a contradictory observation.

Figure 44: Close price vs Moving Averages

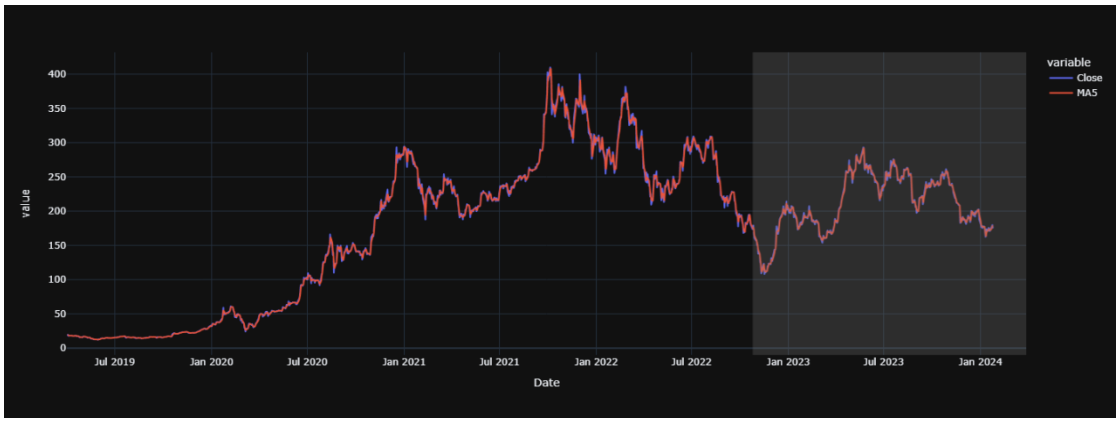

Figure 45: Visual segregation of training and test sets

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Extreme Gradient Boosting	3.6979	80.3085	8.9615	0.9924	0.0552	0.0226

Figure 46: Error metrics on XGBoost for Close price

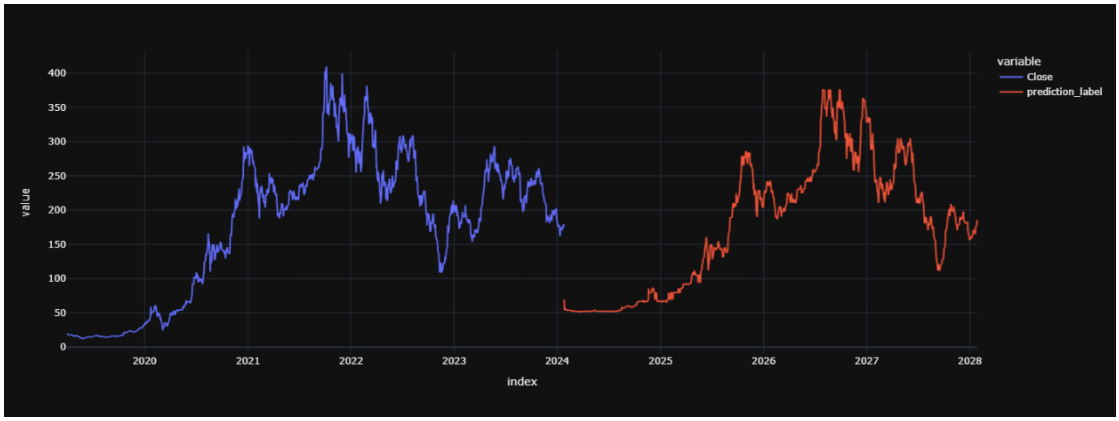

Figure 47: Actual and predicted Close prices

Figure 48: High price vs Moving Averages

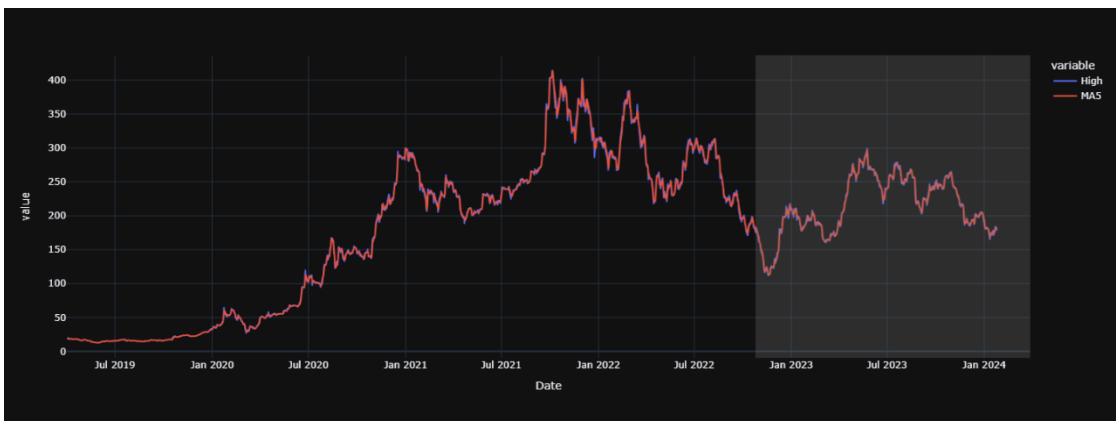

Figure 49: Visual segregation of training and test sets

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Extreme Gradient Boosting	3.0321	64.8813	8.0549	0.9941	0.0497	0.0187

Figure 50: Error metrics on XGBoost for High price

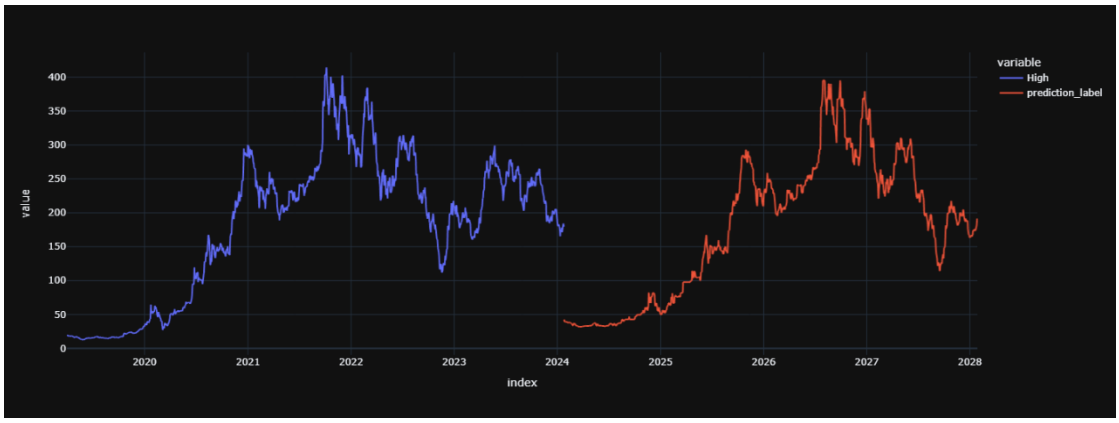

Figure 51: Actual and predicted High prices

Figure 52: Low price vs Moving Averages

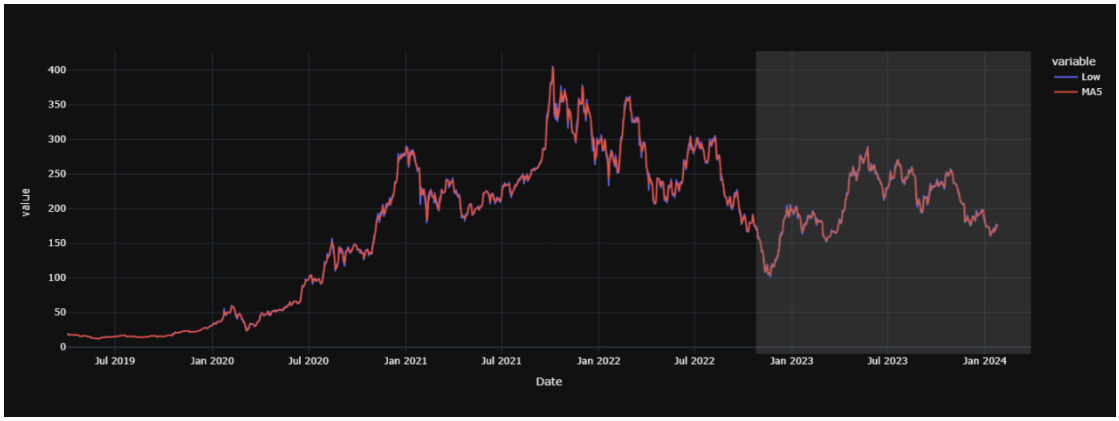

Figure 53: Visual segregation of training and test sets

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Extreme Gradient Boosting	3.4398	54.1061	7.3557	0.9947	0.0457	0.0212

Figure 54: Error metrics on XGBoost for Low price

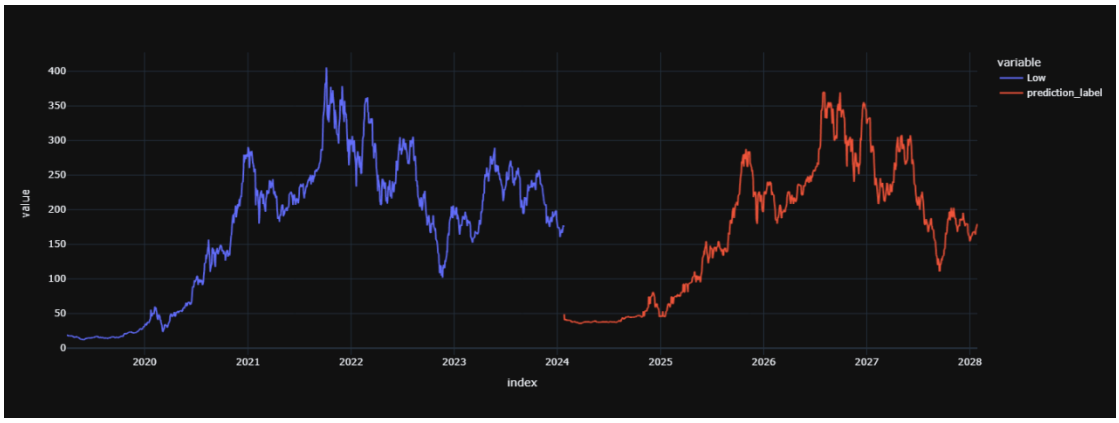

Figure 55: Actual and predicted Low prices

The predictions and error metrics' values for the trading volume present to be an exception to the observations made regarding the other feature in relation to this model. The MSE as depicted in Figure 58 is quite large, while the visual representation of the historical 'Volume' values alongside the predicted values depicts a more stable prediction when compared to the other features. This leads us to conclude that the high value of the MSE might be caused by the large scale of values in the data rather than poor model performance.

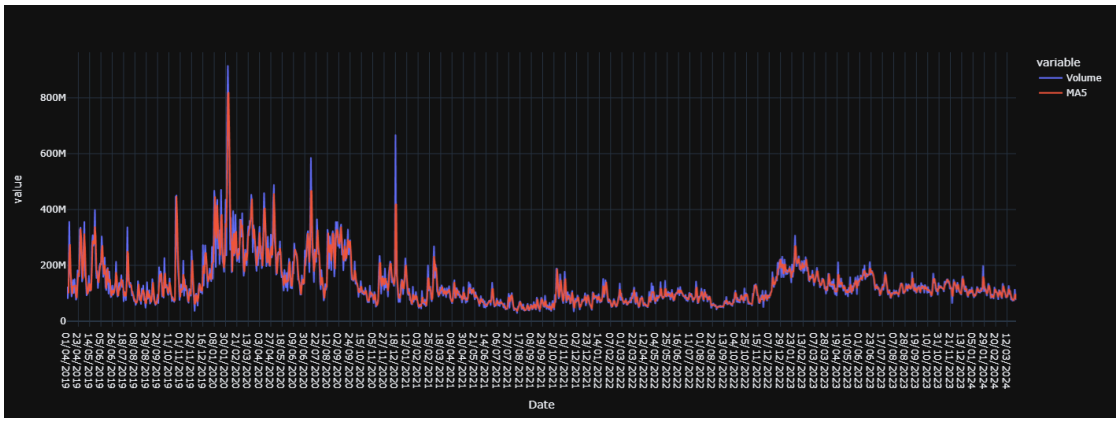

Figure 56: Original Volume vs Moving Averages

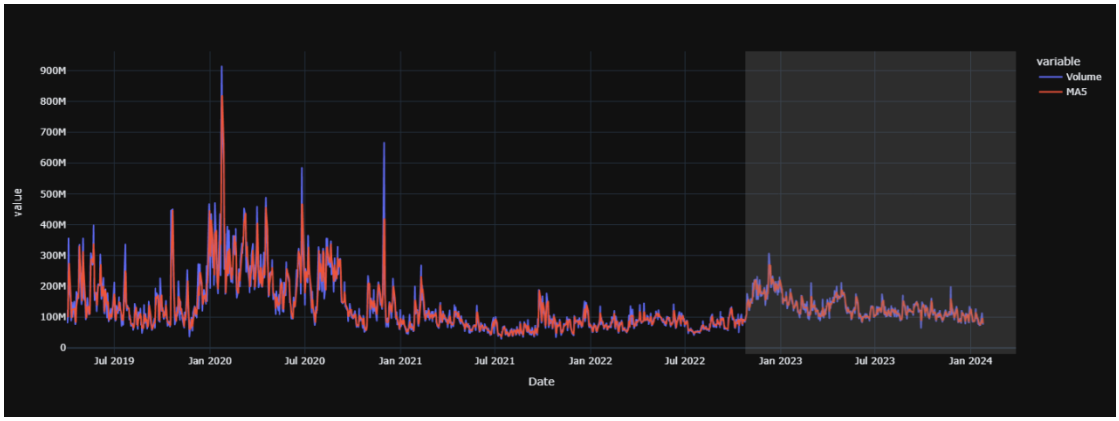

Figure 57: Visual segregation of training and test sets

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Extreme Gradient Boosting	20625473.6661	2130048363797740.2500	46152446.9969	0.7033	0.1830	0.1288

Figure 58: Error metrics on XGBoost for Volume

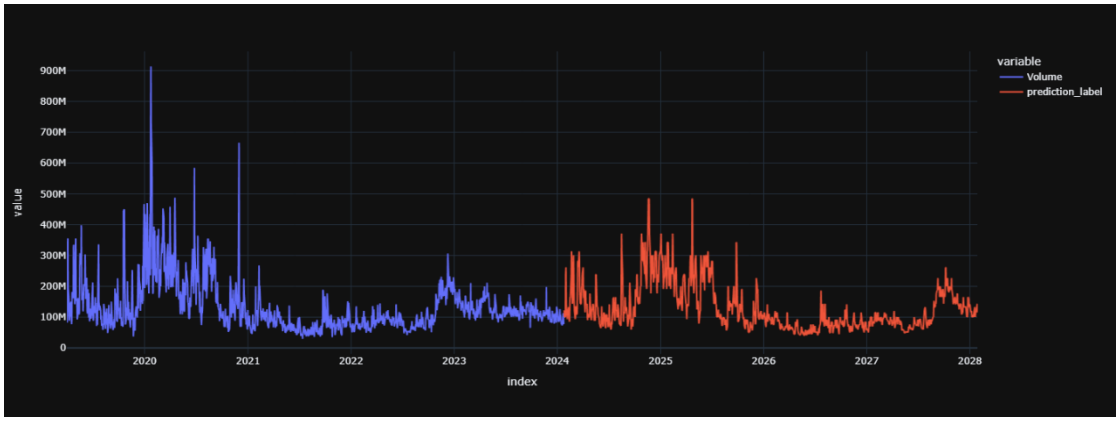

Figure 59: Actual and predicted Volume

In conclusion, the results generated from this model are significantly contradictory and can be inconclusive.

Vector Autoregression

The final model build is the Vector Autoregression (VAR) model with the purpose of creating a model that is capable of capturing complex multivariate non-linear relationships. This model is executed with the help of the elaborate series of steps mentioned before in the implementation of this model.

The first chart, Figure 60, provides a visual representation of all the features in the dataset to help observe the historical values of the features side-by-side. This also helps provide us a more comprehensive understanding of the values when compared with the charts in Figure 61 that illustrates the decomposition of each feature which is further divided into two categories, trend and residuals.

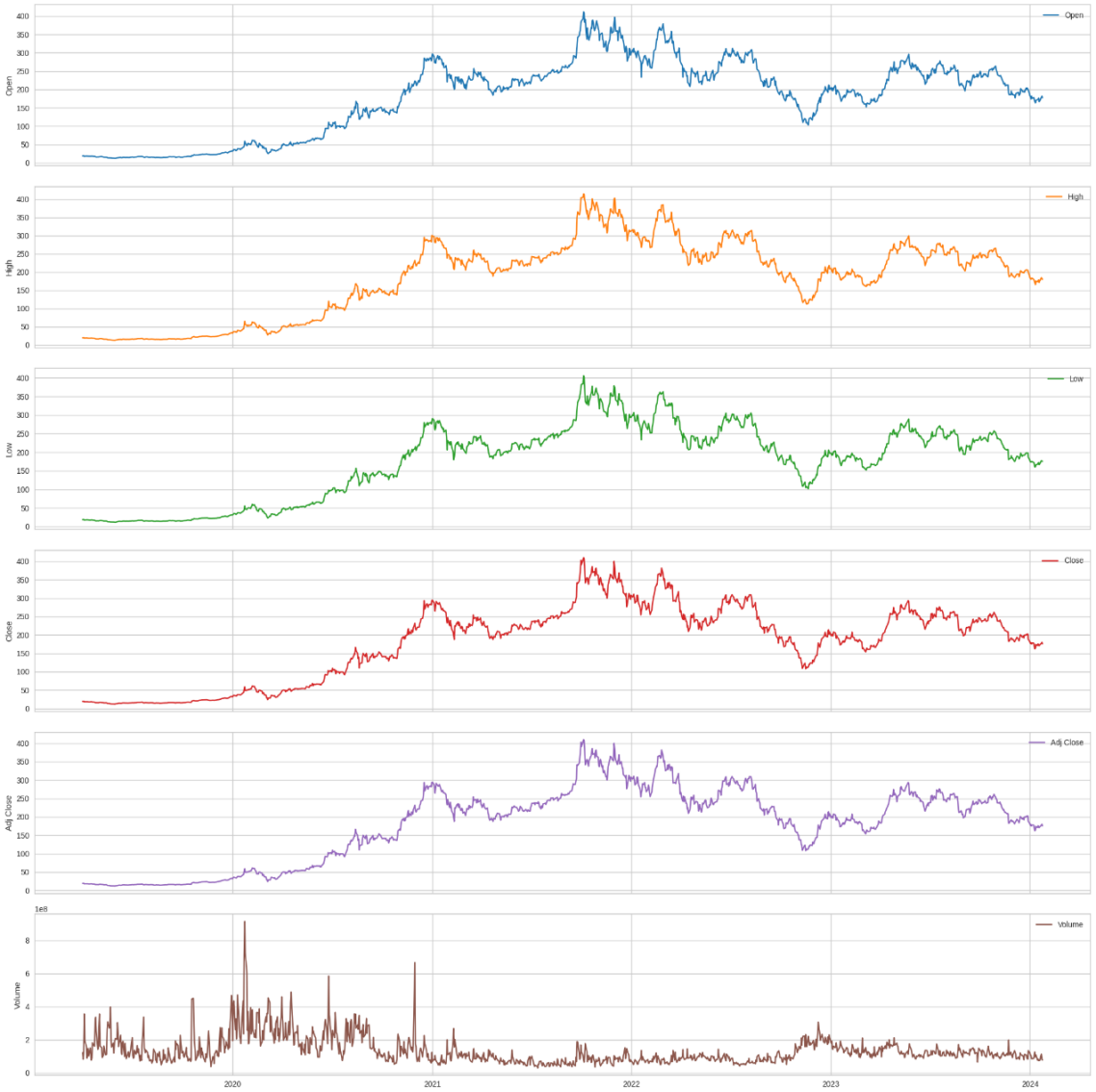

Figure 60: Historical values for all features of the dataset

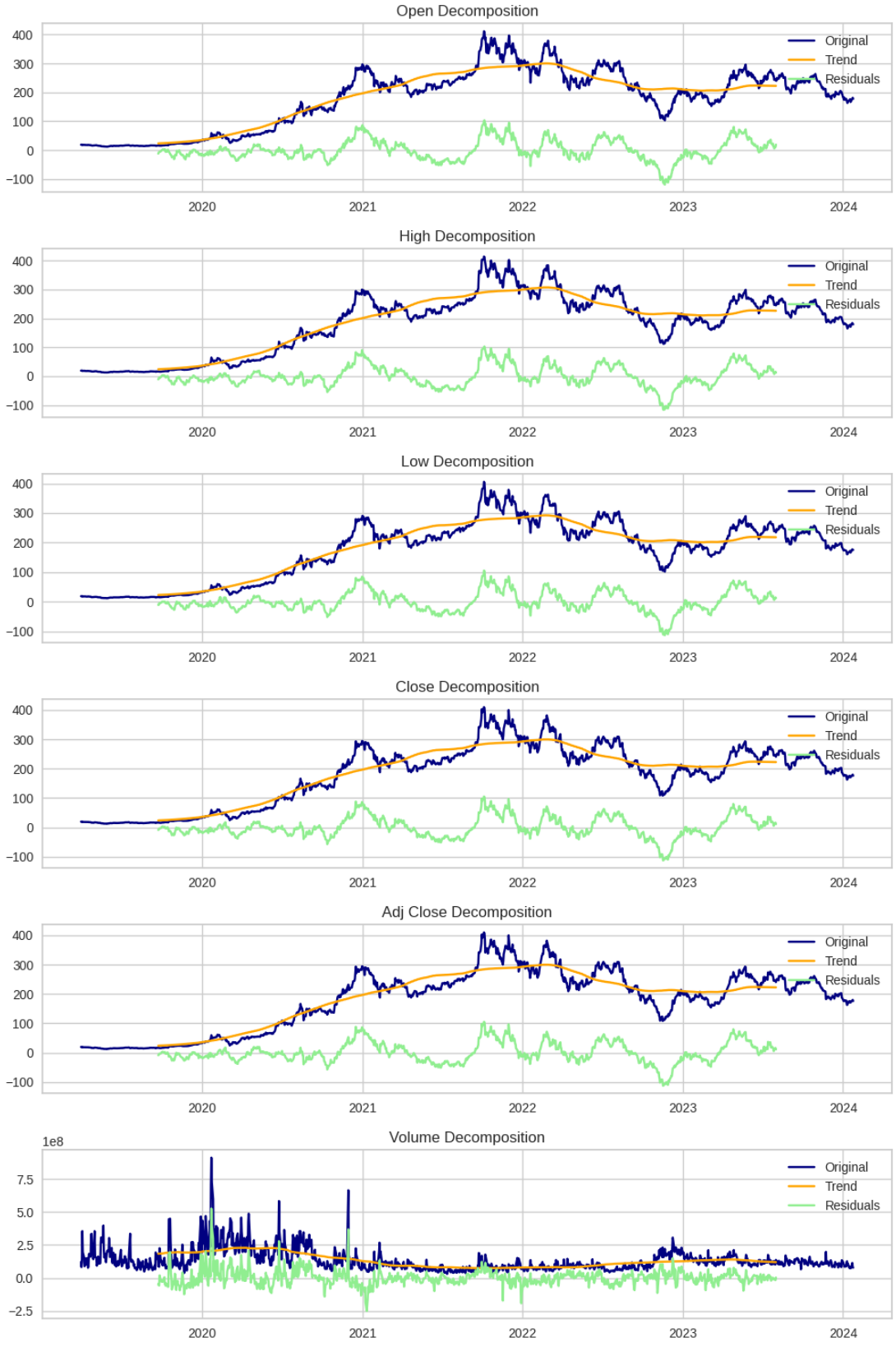

Figure 61: Feature Decomposition

The first execution of the model produces two resultant charts for ‘Open’. The first chart, as depicted in Figure 62, visually represents the historical residual values of open prices and the forecasted residual values, and the second chart, as depicted in Figure 63, represent the historical and predicted values for ‘Open’.

From this, we can observe that the model performs fairly well in capturing the trend and the relationship between ‘Open’ and the other features. This observation can be supported with the use of the MSE value generated by the model for this feature, as shown in Figure 64, which is fairly low.

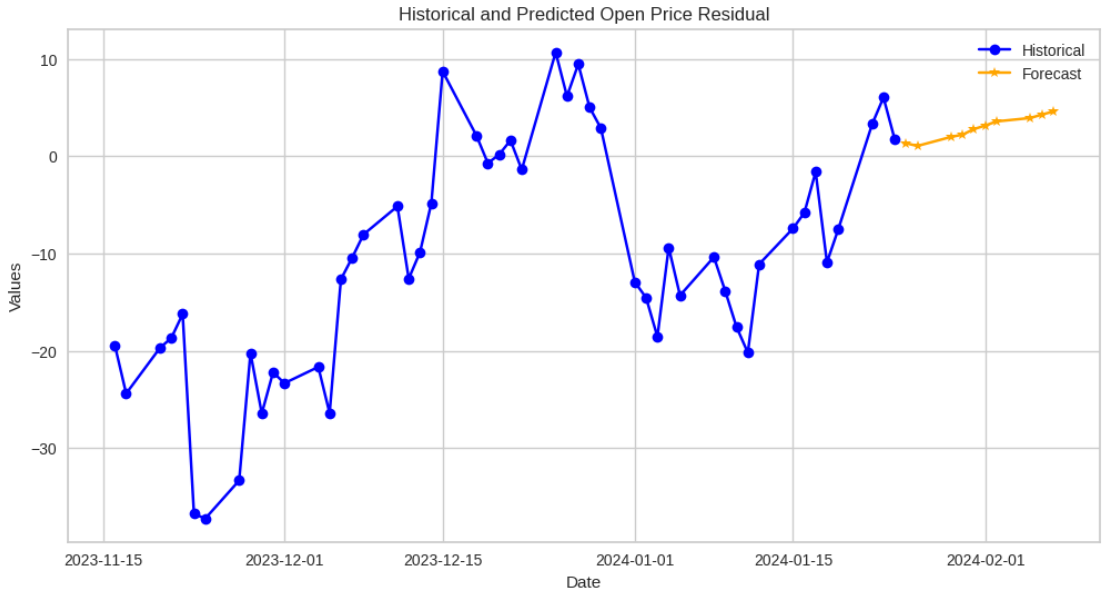

Figure 62: Historical and Predicted residuals for Open

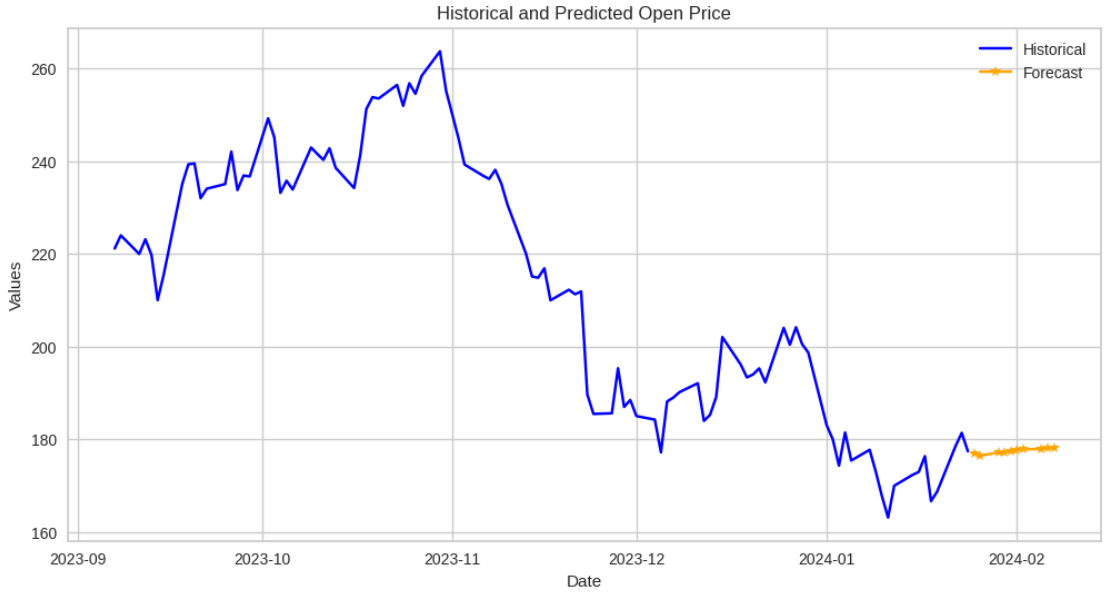

Figure 63: Open price Historical and Forecast

```
Mean Squared Error: 48.98669967128439
```

Figure 64: Open price MSE

A similar conclusion can be made when we observe the resultant charts and MSE values for ‘High’, ‘Low’, ‘Close’, and ‘Volume’, respectively.

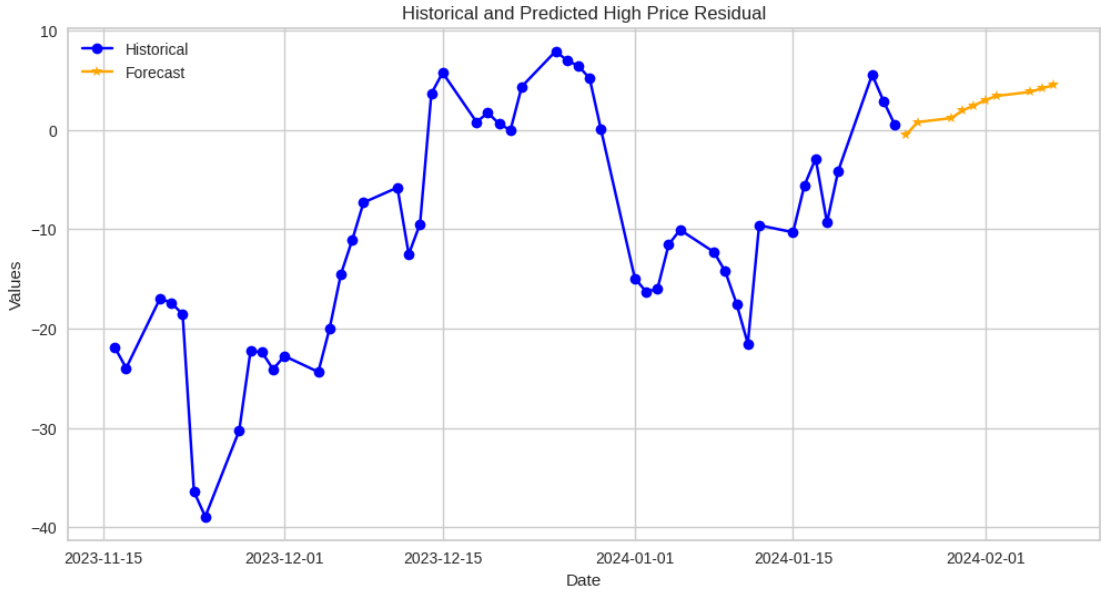

Figure 65: Historical and Predicted residuals for High

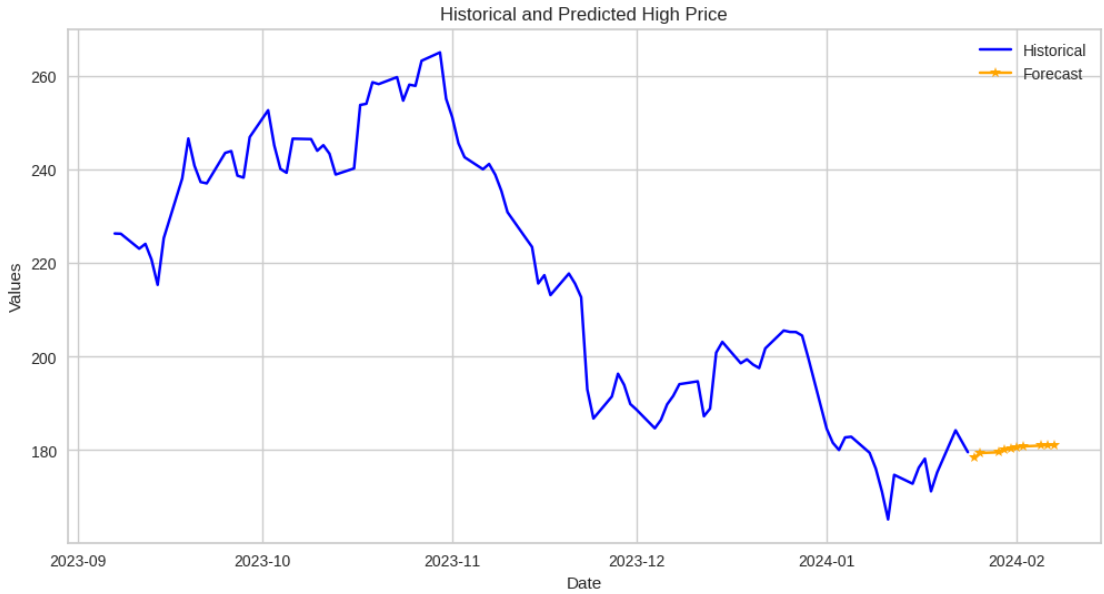

Figure 66: High price Historical and Forecast

```
Mean Squared Error: 39.33771270636573
```

Figure 67: High price MSE

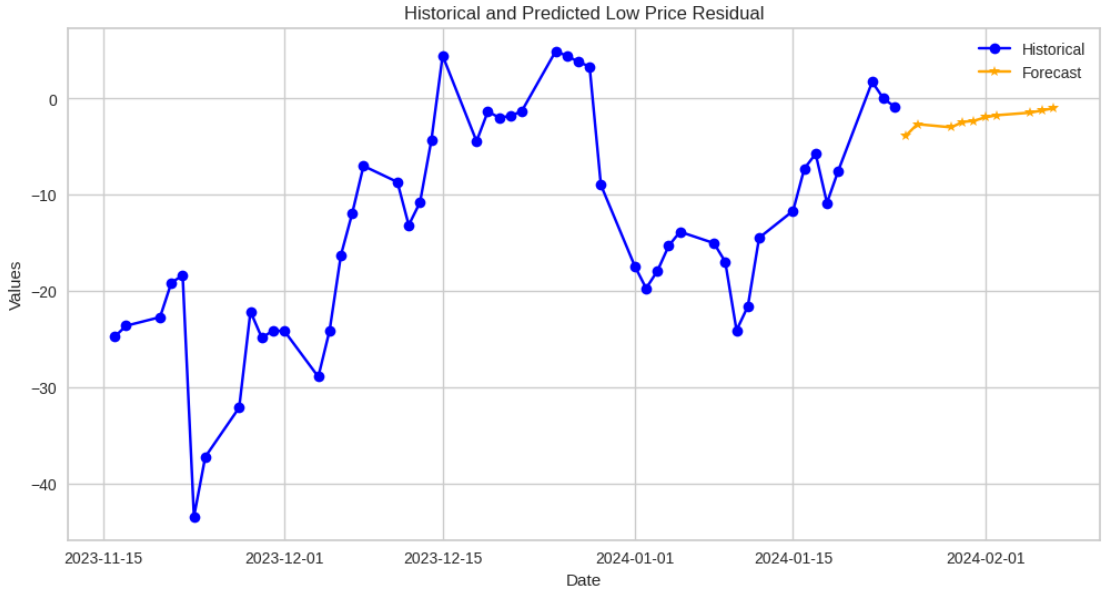

Figure 68: Historical and Predicted residuals for Low

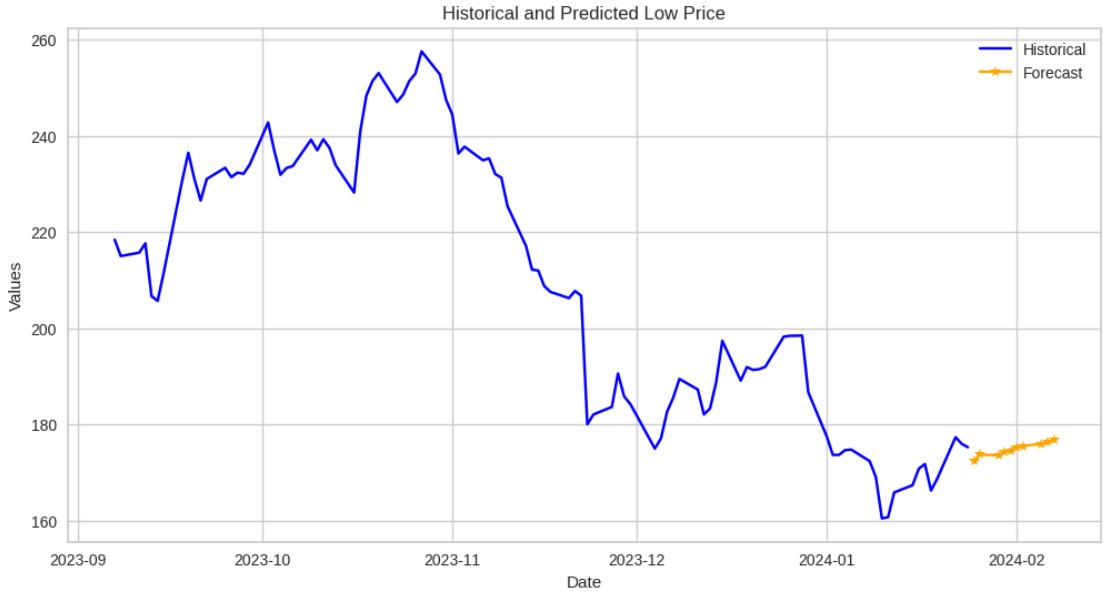

Figure 69: Low price Historical and Forecast

```
Mean Squared Error: 38.657702447991895
```

Figure 70: Low price MSE

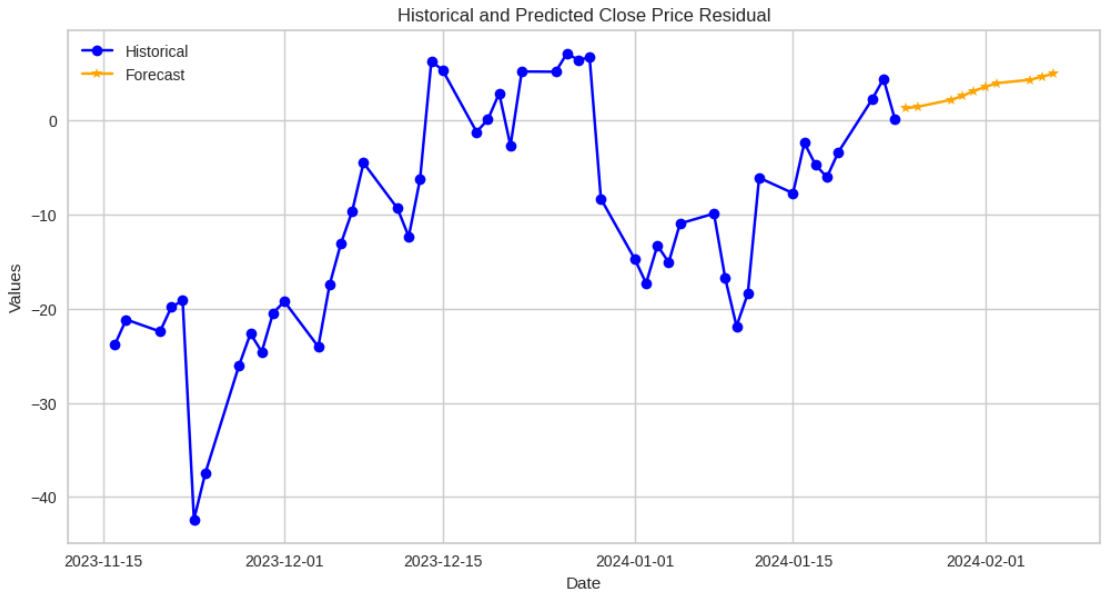

Figure 71: Historical and Predicted residuals for Close

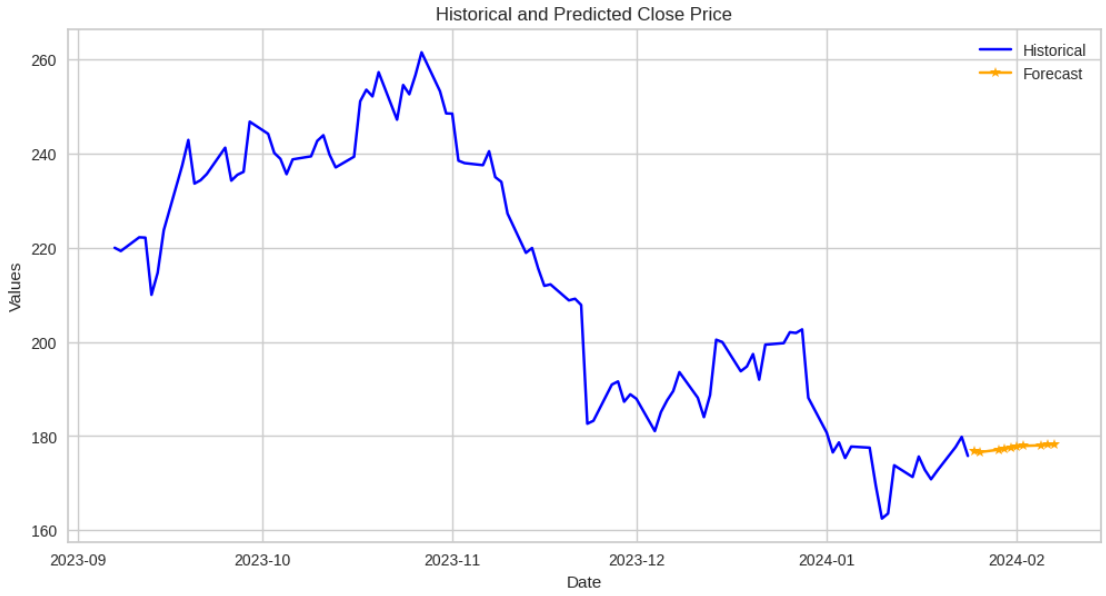

Figure 72: Close price Historical and Forecast

```
Mean Squared Error: 32.342655967677985
```

Figure 73: Close price MSE

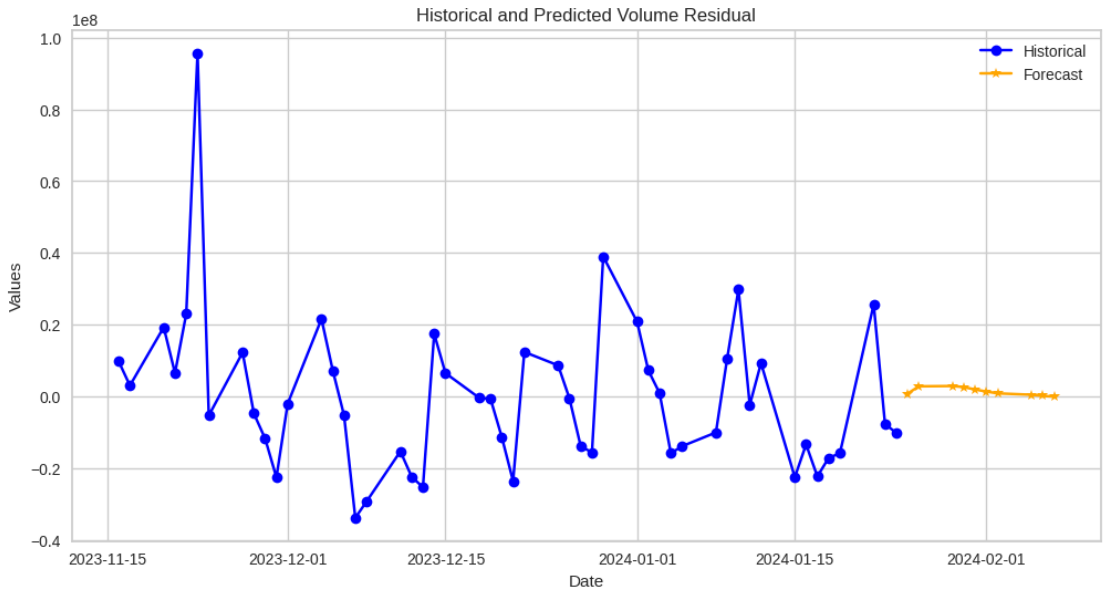

Figure 74: Historical and Predicted residuals for Volume

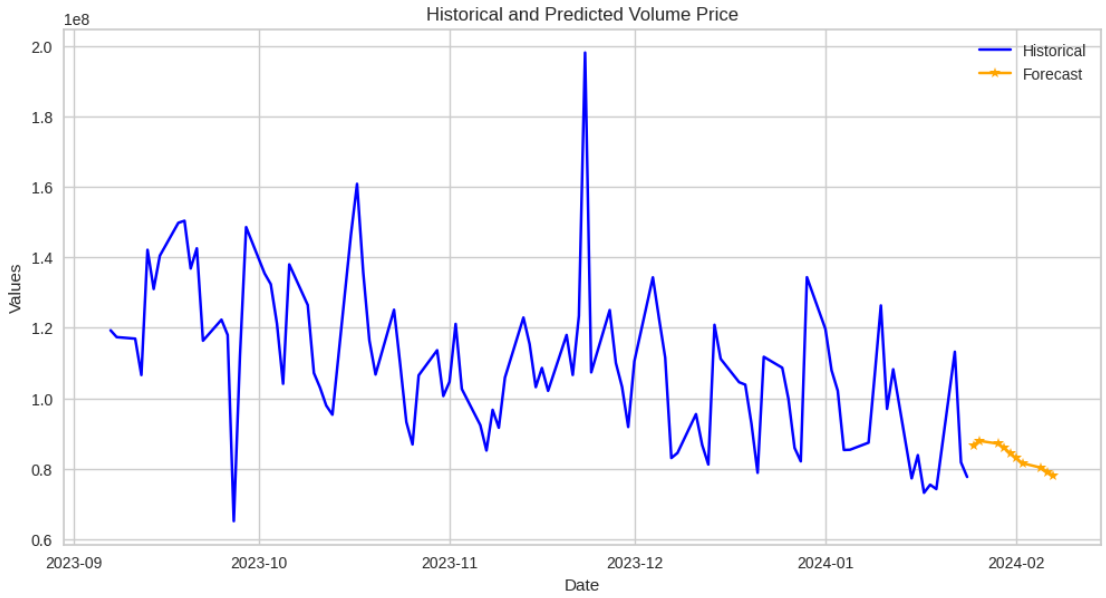

Figure 75: Volume Historical and Forecast

```
Mean Squared Error: 194377069367685.3
```

Figure 76: Volume MSE

Limitations and Challenges:

The models build for the purpose of this study, while performing fairly well, presented with their own sets of limitations and challenges. Some challenges faced, were related to possible shortcomings in data preparation or hyperparameter tuning, while others were related to the nature of the dataset as a whole. These limitations and challenges are discussed below:

- One of the first challenges encountered during the course of this study was the limited amount of information and previous literature that was available with regards to the Support Vector Regression model. Most of the literature was based on the use of SVM and not its extension, SVR, which led to having limited information at hand when building the SVR model.
- The BiLSTM model, while performing significantly well with the other features of the dataset, fell short in relation to the 'Volume' feature. These issues could be fixed by increasing the number of epochs run for this feature, however, the high possibility of overfitting and the possible increment of time complexity, prevented the use of a higher number of epochs for a feature with a vast range of large values.
- The visual representation of the results generated from the SVR model, when considered alongside the MSE, prove to be contradictory and inconclusive.
- Changes in the values of the hyperparameter, gamma, C, and epsilon, for the SVR model, only produced higher values of error. Thus, leading to a conclusion that the model might not be the most suitable of the dataset or alternatively, further normalization and scaling is required.
- The results from the time series model demonstrate that there may be potential overfitting of the model, however, further tuning of the model could conversely lead to a higher MSE.
- When implementing the model for VAR, using 'Volume' as one of the influencing factors for the other features, helped the model perform slightly better. However, it was discovered that with the forecast of future 'Volume', the model not only had a large MSE but the visual representation also illustrated that the resultant forecast might not be as reliable for this feature.

To summarize, overfitting was found to be a persistent challenge which prevented further preprocessing the data. The feature, 'Volume', was performing very well when used with the models as an independent variable. However, when made the target variable with other features as the independent variables, it was found unable to stand dependently. This produced large MSE values. The MSE reduction was not possible due its large-scale values, giving rise to the possibility of either overfitting or a significant increase of time complexity.

7. CONCLUSION

The primary aim of this study was to predict Tesla's stock prices by leveraging advanced machine learning methods and identifying the most effective approach. Various methods were explored, including Bidirectional Long Short-Term Memory (BiLSTM), Support Vector Regression (SVR), time series forecasting with XGBoost, and the Vector Autoregression (VAR) model. After extensive experimentation and analysis, BiLSTM emerged as the best-performing model for predicting Tesla's stock prices.

BiLSTM's ability to capture both past ad future temporal dependencies allowed it to outperform other models. Unlike traditional methods, which often focus only on past data points, BiLSTM's dual processing of input sequences in both forward and backward directions proved highly effective in capturing intricate price patterns and market behaviours. This capability was particularly valuable in modelling long-term dependencies in time series data, which other models like SVR and XGBoost struggled to handle efficiently.

The SVR model, while competent in certain cases, shows limitations in handling the high volatility and complex patterns present in stock market data. Its performance was acceptable for short-term predictions, but it lagged behind BiLSTM in terms of capturing significant price movements and trends over extended periods of time. The model's inability to account for the broader temporal dynamics of stock price data posed a limitation to its accuracy, particularly in volatile market conditions like often seen in Tesla's stock.

The XGBoost model with time series is widely recognized for its high performance in structured data. It faced several challenges when applied to the time series data in this study. Although it was able to capture the trends and patterns in the data, it did not perform as well as BiLSTM because it lacked the inherent mechanism to account for the sequential nature of stock price data. Time series forecasting requires a model to understand the order and temporal correlation between data points. This is something XGBoost struggled with due to its tree-based structure. While effective for static data predictions, XGBoost could not demonstrate the adaptability required for the dynamic and volatile nature of the stock market environment.

The VAR model, which is another time series approach, demonstrated reasonable performance but could not match the accuracy of BiLSTM. While VAR can handle multiple time series and detect relationships among different variables, it runs on the assumption that the data is stationary. This is often not the case with stock market data. Tesla's stock, like most, exhibits trends, volatility, and non-stationary behaviour that limits VAR's applicability. Additionally, VAR's reliance on linear relationships between variables restricts its capability to model the highly non-linear behaviour seen in the stock market.

In contrast, BiLSTM's deep learning architecture allowed it to model non-linear and complex relationships in stock prices efficiently. Its ability to learn from long historical sequences and understand both past and future data gave it an advantage over the more conventional models like SVR, XGBoost,

and VAR. The results demonstrated that BiLSTM captured the major trends, peaks, and turning points more accurately, making it the most reliable model for this forecasting task.

However, while BiLSTM outperformed the other models, there were still limitations to its application. As with most machine learning models, the performance of BiLSTM is highly dependent on the availability and quality of data, as well, as the careful tuning of hyperparameters. Moreover, the model's capability to predict short-term fluctuations remains limited. Stock markets are affected by a vast variety of unpredictable external factors, including news events, political developments, and macroeconomic conditions which are difficult to capture purely from historical price data.

In conclusion, the BiLSTM model proved to be the most robust and accurate model for predicting Tesla's stock prices compared to SVR, XGBoost, and VAR. Its ability to capture long-term dependencies and non-linear relationships made it highly effective in forecasting significant trends. While there are limitations in predicting short-term volatility, BiLSTM's overall performance shows that it is a valuable tool for financial forecasting. Incorporating more external data sources and further optimizing the model could lead to even better results, helping economists, investors, and traders make more informed decisions.

8. FURTHER WORK

While this project has demonstrated the efficacy of the BiLSTM model in predicting Tesla's stock prices, several areas of improvement and future research have emerged. These further work suggestions aim to enhance the model's predictive accuracy, scalability, and applicability to other stocks and markets, ensuring more robust and comprehensive stock forecasting models.

1. Incorporating external data: one of the limitations of the current model is its reliance on historical stock price data alone. In future work, additional data sources such as macroeconomic indicators (such as interest rates, inflation rates), financial news sentiments, or social media data could be incorporated into the model. By analysing broader economic and market trends, the model would be better equipped to predict price movements influenced by external factors. NLP techniques could be integrated into the model to analyse news articles, earnings reports, and social media sentiments, providing a more comprehensive overview of the factors influencing Tesla's stock prices.
2. Model optimization: while the BiLSTM model performed well, there is still room for optimizing the architecture. Future work could involve experimenting with more sophisticated deep learning models, such as Attention mechanisms or Transformer-based architectures, which have also proven highly effective in various time series prediction tasks. Such models are better suited for capturing long-range dependencies and assigning varying importance to different time steps in input sequences, potentially leading to improved performance.
3. Hyperparameter tuning: the current model underwent hyperparameter tuning, however, further refinement is possible. Future research could employ more advanced optimization algorithms such as Bayesian optimization or grid search techniques to find the optimal set of hyperparameters.
4. Ensemble methods: future work could explore combining the BiLSTM model with other models, such as SVMs, Decision Tress, or even traditional methods like ARIMA. Ensemble methods, which aggregate predictions from multiple models, are known to improve robustness and accuracy by compensating for the weaknesses of the individual models. By leveraging both machine learning and statistical approaches, a hybrid model could achieve better performance in stock price prediction.
5. Real-Time Stock prediction: implementing the BiLSTM model for real-time stock prediction would be an exciting future direction. This would involve adapting the model to make predictions on streaming data, enabling it to provide near-instantaneous forecasts as new stock price data becomes available. To support real-time processing, the model's computational efficiency would need to be optimized, possibly by developing it in a cloud environment using services like TensorFlow Serving or AWS SageMaker.
6. Application to other stocks and markets: Although the model performed well on Tesla's stock prices, it would be valuable to test the BiLSTM model on a broader spectrum of stocks across

different industries and markets. Each stock exhibits unique price behaviours and volatility patterns, and applying the model to various stocks could help assess its generalizability. Additionally, expanding the scope to international markets and incorporating forex or commodity data could provide more insights into the model's versatility.

7. Risk Management and Portfolio Optimization: beyond simply predicting stock prices, future work could focus on integrating the BiLSTM model into a broader financial strategy for risk management and portfolio optimization. By predicting price movements, the model could assist in developing trading strategies that minimize risk and maximize returns. Moreover, by using reinforcement learning techniques, future models could be trained to make investment decisions dynamically, adjusting a portfolio in response to predicted price changes.

In conclusion, the BiLSTM model provides a promising foundation for stock price prediction, but further enhancements and expansions could make it even more effective. Through the integration of external data sources, optimization of model architecture, and real-time applications, future work could lead to more accurate and comprehensive financial forecasting tools.

9. REFERENCES

Abdullah, L.T., 2022. Forecasting time series using Vector Autoregressive Model. Int. J. Nonlinear Anal. Appl. 13, 499–511. https://doi.org/10.22075/ijnaa.2022.5521

Bai, Y., Bai, P., Zhang, X., Li, H., 2023. Financial Asset Volatility Forecasting using LSTM with Intraday High-Low Price Information. Presented at the Proceedings of the 2nd International Conference on Mathematical Statistics and Economic Analysis, MSEA 2023, May 26–28, 2023, Nanjing, China.

Comito, C., Pizzuti, C., 2022. Artificial intelligence for forecasting and diagnosing COVID-19 pandemic: A focused review. Artif. Intell. Med. 128, 102286. https://doi.org/10.1016/j.artmed.2022.102286

F., A., Elsir, S., Faris, H., 2015. A Comparison between Regression, Artificial Neural Networks and Support Vector Machines for Predicting Stock Market Index. Int. J. Adv. Res. Artif. Intell. 4. https://doi.org/10.14569/IJARAI.2015.040710

Kim, K., 2003. Financial time series forecasting using support vector machines. Neurocomputing, Support Vector Machines 55, 307–319. https://doi.org/10.1016/S0925-2312(03)00372-2

Rokhsatyazdi, E., Rahnamayan, S., Amirinia, H., Ahmed, S., 2020. Optimizing LSTM Based Network For Forecasting Stock Market, in: 2020 IEEE Congress on Evolutionary Computation (CEC). Presented at the 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–7. https://doi.org/10.1109/CEC48606.2020.9185545

Selvin, S., Vinayakumar, R., Gopalakrishnan, E.A., Menon, V.K., Soman, K.P., 2017. Stock price prediction using LSTM, RNN and CNN-sliding window model, in: 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Presented at the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1643–1647. https://doi.org/10.1109/ICACCI.2017.8126078

Siami-Namini, S., Tavakoli, N., Namin, A.S., 2019. A Comparative Analysis of Forecasting Financial Time Series Using ARIMA, LSTM, and BiLSTM. https://doi.org/10.48550/arXiv.1911.09512

Simkins, S., 1995. Forecasting with vector autoregressive (VAR) models subject to business cycle restrictions. Int. J. Forecast. 11, 569–583. https://doi.org/10.1016/0169-2070(95)00616-8

Vuong, P.H., Dat, T.T., Mai, T.K., Uyen, P.H., Bao, P.T., 2022. Stock-Price Forecasting Based on XGBoost and LSTM. | Computer Systems Science & Engineering | EBSCOhost [WWW Document]. https://doi.org/10.32604/csse.2022.017685

Zaheer, S., Anjum, N., Hussain, S., Algarni, A.D., Iqbal, J., Bourouis, S., Ullah, S.S., 2023. A Multi Parameter Forecasting for Stock Time Series Data Using LSTM and Deep Learning Model. Mathematics 11, 590. https://doi.org/10.3390/math11030590

10. APPENDIX

Appendix 1: Code lines for Data Description and Visualization

- Code 1: Data description

```
description = data.describe()

print(description)
```

- Code 2: Codes for data visualization

```
from pylab import rcParams
```

```
rcParams['figure.figsize'] = 14, 8
sns.set(style='whitegrid', palette='muted', font_scale=1)

ax = data.plot(x="Date", y="Open");
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('Tesla Daily Open Price')
```

```
ax = data.plot(x="Date", y="High");
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('Tesla Daily High Price')
```

```
ax = data.plot(x="Date", y="Low");
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('Tesla Daily Low Price')
```

```
ax = data.plot(x="Date", y="Close");
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('Tesla Daily Close Price')
```

```
ax = data.plot(x="Date", y="Adj Close");
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('Tesla Daily Adjusted Close Price')
```

```
ax = data.plot(x="Date", y="Volume");
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('Tesla Daily Volume')
```

```
ax = data.plot(x="Date", y=["Open", "Close"]);
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('Tesla Daily Open Price vs Close Price')
```

```
ax = data.plot(x="Date", y=["High", "Low"]);
ax.set_xlabel('Date')
ax.set_ylabel('Price ($)')
ax.set_title('Tesla Daily High Price vs Low Price')
```

Appendix 2: Code for MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
open_price = data.Open.values.reshape(-1, 1)
scaled_open = scaler.fit_transform(open_price)

close_price = data.Close.values.reshape(-1, 1)
scaled_close = scaler.fit_transform(close_price)

high_price = data.High.values.reshape(-1, 1)
scaled_high = scaler.fit_transform(high_price)

low_price = data.Low.values.reshape(-1, 1)
scaled_low = scaler.fit_transform(low_price)

vol = data.Volume.values.reshape(-1, 1)
scaled_vol = scaler.fit_transform(vol)
```

Appendix 3: Code lines for implementing BiLSTM model

- Code 1: Importing TensorFlow and Keras for BiLSTM

```
from tensorflow import keras
from tensorflow.keras.layers import Bidirectional, Dropout, Activation, Dense, LSTM
from tensorflow.keras.models import Sequential
```

- Code 2: Sequences and train and test split

```
seq_length = 120
def splitseq(df, seq_length):
  n_seq = len(df) - seq_length + 1
  return np.array([df[i:(i+seq_length)] for i in range(n_seq)])

def train_test_sets(df, seq_length, train_frc):
  sequences = splitseq(df, seq_length)
  n_train = int(sequences.shape[0] * train_frc)
  x_train = sequences[:n_train, :-1, :]
  y_train = sequences[:n_train, -1, :]
  x_test = sequences[n_train:, :-1, :]
  y_test = sequences[n_train:, -1, :]
  return x_train, y_train, x_test, y_test

X_train, y_train, X_test, y_test = train_test_sets(scaled_open, seq_length, train_frc=0.8)
```

- Code 3: Building the layers of the LSTM model

```
dropout = 0.2
window_size = seq_length - 1

model = keras.Sequential()

model.add(LSTM(window_size, return_sequences=True, input_shape=(window_size, X_train.shape[-1])))

model.add(Dropout(rate=dropout))
model.add(Bidirectional(LSTM(window_size * 2, return_sequences=True)))

model.add(Dropout(rate=dropout))
model.add(Bidirectional(LSTM(window_size, return_sequences=False)))

model.add(Dense(units=1))
model.add(Activation('linear'))
```

- Code 4: Training the LSTM model

```
batch_size = 16
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(X_train, y_train, epochs=20, batch_size=batch_size, shuffle=False, validation_split=0.2)
```

- Code 5: Predicting and Visualizing predicted values against actual value from LSTM

```
y_pred = model.predict(X_test)

y_test_org = scaler.inverse_transform(y_test)
y_pred_org = scaler.inverse_transform(y_pred)

plt.plot(y_test_org, label='Actual Price', color='blue')
plt.plot(y_pred_org, label='Predicted Price', color='green')

plt.title('Tesla Open Price Prediction')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend(loc='best')

plt.show()
```

- Code 6: Visualizing changes in MSE over 30 epochs

```
plt.plot(history.history['loss'], label='Training MSE')
plt.plot(history.history['val_loss'], label='Validation MSE')
plt.title('Model Mean Squared Error')
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
plt.show()
```

Appendix 4: Code lines for implementing SVR model

- Code 1: Importing SVR and MSE

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

- Code 2: Setting up SVR model

```
model = SVR(kernel='rbf', gamma=0.5, C=10, epsilon=0.05)
```

- Code 3: Manual training and test split

```
open_train = scaled_open[:900]
open_test = scaled_open[900:]
close_train = scaled_close[:900]
close_test = scaled_close[900:]
high_train = scaled_high[:900]
high_test = scaled_high[900:]
low_train = scaled_low[:900]
low_test = scaled_low[900:]
volume_train = scaled_vol[:900]
volume_test = scaled_vol[900:]
```

- Code 4: Dataset sequence creation

```
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        v = X[i:i + time_steps]
        Xs.append(v)
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

TIME_STEPS = 5
X_train, y_train = create_dataset(open_train, open_train, TIME_STEPS)
X_test, y_test = create_dataset(open_test, open_test, TIME_STEPS)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(895, 5, 1) (895, 1)
(353, 5, 1) (353, 1)
```

- Code 5: SVR model training

```
model.fit(X_train.reshape(X_train.shape[0], -1), y_train)
```

- Code 6: Predicting using trained model

```
train_pred = model.predict(X_train.reshape(X_train.shape[0], -1))
test_pred = model.predict(X_test.reshape(X_test.shape[0], -1))

train_pred_inv = scaler.inverse_transform(train_pred.reshape(-1, 1))
y_train_inv = scaler.inverse_transform(y_train.reshape(-1, 1))
test_pred_inv = scaler.inverse_transform(test_pred.reshape(-1, 1))
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
```

- Code 7: Calculating the mean squared error for SVR

```
mse_train = mean_squared_error(y_train_inv, train_pred_inv)
mse_test = mean_squared_error(y_test_inv, test_pred_inv)
print("Mean Squared Error on Training Data:", mse_train)
print("Mean Squared Error on Testing Data:", mse_test)
```

- Code 8: Visualizing the actual and predicted values from SVR

```
plt.figure(figsize=(15, 8))
plt.plot(data.index[TIME_STEPS:TIME_STEPS+len(train_pred_inv)], y_train_inv, label='Original', color='darkgreen')
plt.plot(data.index[TIME_STEPS:TIME_STEPS+len(train_pred_inv)], train_pred_inv, label='Predicted', color='red')
plt.xlabel('Date')
plt.ylabel('Open Price ($)')
plt.title('Tesla Open Price Forecasting with SVR')
plt.legend()
plt.show()
```

```
y_test_inv = y_test_inv.flatten()

x_values = data.index[TIME_STEPS+len(train_pred_inv)+TIME_STEPS-1 : TIME_STEPS+len(train_pred_inv)+TIME_STEPS-1 + len(y_test_inv)]

plt.figure(figsize=(15, 8))
plt.plot(x_values, y_test_inv, label='Original', color='darkgreen')
plt.plot(x_values, test_pred_inv, label='Predicted', color='red')
plt.xlabel('Date')
plt.ylabel('Open Price ($)')
plt.title('Tesla Open Price Forecasting with SVR - Test Dataset')
plt.legend()
plt.show()
```

Appendix 5: Code lines for Time Series with XGBoost

- Code 1: Installing PyCaret

```
!pip install pycaret
```

- Code 2: Data preparation

```
# create 5 year moving average
df['MA5'] = df['Open'].rolling(2).mean()
# plot the df and MA
import plotly.express as px
fig = px.line(df, x="Date", y=["Open", "MA5"], template = 'plotly_dark')
fig.show()
```

- Code 3: Creating a Series and Data Formatting

```
# create a sequence of numbers
df['Series'] = np.arange(1,len(df)+1)
# drop unnecessary columns and re-arrange
df.drop(['Close','High','Low','Adj Close','Volume'],axis=1, inplace=True)
df = df[['Series','Date','Open','MA5']]
# check the head of the dfset
df.head()
```

- Code 4: Date Formatting

```
df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
df['Date'] = df['Date'].dt.strftime('%d%m%Y').astype(int)
```

- Code 5: Train-Test Split

```
# split data into train-test set manually
train = df[:900]
test = df[900:]
# check shape
train.shape, test.shape
```

- Code 6: PyCaret Setup and Model training

```
# import the regression module
from pycaret.regression import *
# initialize setup
s = setup(data=train, test_data=test, target='Open', fold_strategy='timeseries', numeric_features=['Date', 'Series'],
          fold=3, transform_target=True, fold_shuffle=False, data_split_shuffle=False, session_id=123)
```

- Code 7: Model creation using XGBoost

```
best = create_model('xgboost')
```

- Code 8: Predictions on dataset and Visualization of Training and Test sets

```
# generate predictions on the original dataset
predictions = predict_model(best, data=df)
# add a date column in the dataset
predictions['Date'] = pd.date_range(start='2019-04-01', periods=len(predictions), freq = 'B')
# line plot
fig = px.line(predictions, x='Date', y=["Open","MA5"], template = 'plotly_dark')
# add a vertical rectange for test-set separation
fig.add_vrect(x0="2022-10-25", x1="2024-03-28", fillcolor="grey", opacity=0.25, line_width=0)
fig.show()
```

- Code 9: Finalizing best model and making predictions for the future dataset

```
final_best = finalize_model(best)
```

```
future_dates = pd.date_range(start = '2024-03-28', end = '2028-03-31', freq = 'B')
future_df = pd.DataFrame()
future_df['Date'] = pd.date_range(start = '2024-03-28', end = '2028-03-31', freq = 'B')
future_df['Series'] = np.arange(2000,(2000+len(future_dates)))
future_df.head()
```

```
future_df['MA5'] = df['Open'].rolling(2).mean()
```

```
future_df['Date'] = future_df['Date'].dt.strftime('%d%m%Y').astype(int)
```

```
predictions_future = predict_model(final_best, data=future_df)
predictions_future.head()
```

- Code 10: Visualizing historical and future predicted prices

```
concat_df = pd.concat([df,predictions_future], axis=0)
concat_df_i = pd.date_range(start='2019-04-01', periods=len(concat_df), freq='B')
concat_df.set_index(concat_df_i, inplace=True)
fig = px.line(concat_df, x=concat_df.index, y=["Open", "prediction_label"], template = 'plotly_dark')
fig.show()
```

Appendix 6: Code line for implementing VAR model

- Code 1: Importing ADF test

```
from statsmodels.tsa.stattools import adfuller
```

- Code 2: Setting 'Date' as index

```
df = pd.DataFrame(data)
df.set_index('Date', inplace=True)
df.index = pd.date_range(start='2019-04-01', periods=len(df), freq='B')
```

- Code 3: Performing ADF test on the dataset

```
#Augmented Dickey-Fuller Test on all the columns

def adf_test(series, columns):
  result = adfuller(series)
  print(f'-- ADF Test for {columns} --')
  print('ADF Statistic: %f' % result[0])
  print('p-value: %f' % result[1])
  print('Critical values:')
  for key, value in result[4].items():
    print(f'\t{key}: {value:.2f}')
  if result[1] <= 0.05:
    print(f"Stationary\n")
  else:
    print(f"Non-Stationary\n")

for column in df.columns:
  adf_test(df[column], column)
```

- Code 4: Performing Holt's Exponential Smoothing

```
#Handling non-stationary time series data for VAR modelling

from statsmodels.tsa.holtwinters import Holt

open_price = df['Open']
high_price = df['High']
low_price = df['Low']
close_price = df['Close']
adjclose_price = df['Adj Close']
volume = df['Volume']

open_price_model = Holt(df['Open'])
open_price_results = open_price_model.fit(smoothing_level=0.1)
open_price_trend = open_price_results.fittedvalues
open_price_level = open_price_results.level
open_price_residual = df['Open'] - open_price_trend

high_price_model = Holt(df['High'])
high_price_results = high_price_model.fit(smoothing_level=0.1)
high_price_trend = high_price_results.fittedvalues
high_price_level = high_price_results.level
high_price_residual = df['High'] - high_price_trend

low_price_model = Holt(df['Low'])
low_price_results = low_price_model.fit(smoothing_level=0.1)
low_price_trend = low_price_results.fittedvalues
low_price_level = low_price_results.level
low_price_residual = df['Low'] - low_price_trend
```

```
close_price_model = Holt(df['Close'])
close_price_results = close_price_model.fit(smoothing_level=0.1)
close_price_trend = close_price_results.fittedvalues
close_price_level = close_price_results.level
close_price_residual = df['Close'] - close_price_trend

adjclose_price_model = Holt(df['Adj Close'])
adjclose_price_results = adjclose_price_model.fit(smoothing_level=0.1)
adjclose_price_trend = adjclose_price_results.fittedvalues
adjclose_price_level = adjclose_price_results.level
adjclose_price_residual = df['Adj Close'] - adjclose_price_trend

volume_model = Holt(df['Volume'])
volume_results = volume_model.fit(smoothing_level=0.1)
volume_trend = volume_results.fittedvalues
volume_level = volume_results.level
volume_residual = df['Volume'] - volume_trend
```

- Code 5: Checking seasonal decomposition

```
#Checking for decomposition in the components using Seasonal Decomposition
from statsmodels.tsa.seasonal import seasonal_decompose

columns = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(10,15))

#Performing decomposition
for i, column in enumerate(columns):
  decomposition = seasonal_decompose(df[column], model='additive', period=252)
  #plotting original, trend, and residuals
  axes[i].plot(df.index,df[column], label='Original', color='navy')
  axes[i].plot(decomposition.trend, label='Trend', color='orange')
  axes[i].plot(decomposition.resid, label='Residuals', color='lightgreen')

  axes[i].set_title(f'{column.replace("_", " ").title()} Decomposition')
  axes[i].legend(loc='upper right')

plt.tight_layout()
plt.show()
```

- Code 6: Multicollinearity checking

```
from statsmodels.tsa.api import VAR
from statsmodels.stats.outliers_influence import variance_inflation_factor

data_for_var = pd.concat([df[['High', 'Low', 'Close', 'Volume']],
                          open_price_residual], axis=1, join='inner')

# Calculate VIF to check for multicollinearity
vif_data = pd.DataFrame()
vif_data["Variable"] = data_for_var.columns
vif_data["VIF"] = [variance_inflation_factor(data_for_var.values, i) for i in range(data_for_var.shape[1])]
print(vif_data)



model = VAR(data_for_var)

lag_results = model.select_order(maxlags=10)
print(lag_results.summary())
```

- Code 7: Model fitting

```
# Assuming 'data_for_var' contains the relevant time series data
model = VAR(data_for_var)

# Fitting the model with the chosen lag order
model_fitted = model.fit(2, ic='aic', trend='ct')
```

- Code 8: Calculating Durbin-Watson statistics

```
from statsmodels.stats.stattools import durbin_watson

# Compute Durbin-Watson statistics
dw_stats = durbin_watson(model_fitted.resid)

for col, stat in zip(data_for_var.columns, dw_stats):
    print(f'{col}: {stat}')
```

- Code 9: Forecasting residual values for 10 future steps

```
# Forecasting
forecast_steps = 10  # Number of steps to forecast ahead
forecast = model_fitted.forecast(model_fitted.endog, steps=forecast_steps)

forecast_df = pd.DataFrame(forecast, index=pd.date_range(start=data_for_var.index[-1] + pd.Timedelta(days=1),
                                                         periods=forecast_steps, freq='B'), columns=data_for_var.columns)

# Concatenate the predicted and last 50 observations of historical data for 'open_price_residual'
concatenated_data = pd.concat([data_for_var[0].tail(50), forecast_df[0]])

# Plot the concatenated data
plt.figure(figsize=(12, 6))
plt.plot(data_for_var.index[-50:], data_for_var[0].tail(50), label='Historical', color='blue',marker = 'o')
plt.plot(forecast_df.index, forecast_df[0], label='Forecast', color='orange',marker = '*')
plt.title('Historical and Predicted Open Price Residual')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()
```

- Code 10: Forecasting values for next 10 steps (days)

```
open_price = df['Open']

# the fitted Holt model
open_price_model = Holt(df['Open'])
open_price_results = open_price_model.fit(smoothing_level=0.1)
open_price_trend = open_price_results.fittedvalues
open_price_level = open_price_results.level
open_price_residual = df['Open'] - open_price_trend


# Forecasting the trend component
forecast_steps = 10  # Number of steps to forecast ahead

# Forecasting future trend values
open_price_trend_forecast = open_price_results.forecast(steps=forecast_steps)

# Creating a date range for the forecast
open_price_trend_forecast_index = pd.date_range(start=open_price_trend.index[-1] + pd.Timedelta(days=1),
                                                periods=forecast_steps, freq='B')

# Creating a DataFrame for the forecasted trend
open_price_trend_forecast_df = pd.DataFrame(open_price_trend_forecast,
                                            index=open_price_trend_forecast_index, columns=['open_price_trend_forecast'])

# sum the predicted trend and predicted residual values to get the final forecast
forecast_df['open_price_residual_final'] = forecast_df[0] + open_price_trend_forecast_df['open_price_trend_forecast']
```

```
# Plotting the final forecast of original open_price
plt.figure(figsize=(12, 6))
plt.plot(open_price[-100:], label='Historical', color='blue')
plt.plot(forecast_df.index, forecast_df['open_price_residual_final'], label='Forecast', color='orange', marker='*')
plt.title('Historical and Predicted Open Price')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()
```

- Code 11: Calculating MSE

```
from sklearn.metrics import mean_squared_error

# Get the historical open prices for the last 10 days for comparison
actual_values = open_price[-10:]

# Get the corresponding forecasted open prices
predicted_values = forecast_df['open_price_residual_final']

# Calculate the Mean Squared Error (MSE)
mse = mean_squared_error(actual_values, predicted_values)

print(f'Mean Squared Error: {mse}')
```