Stock Price Prediction Project Report

I. Introduction

Machine learning has revolutionized various industries, including the financial market, by enabling accurate prediction of stock prices. In this capstone project, we focus on predicting the stock prices of Google, a leading tech giant. By leveraging advanced machine learning techniques, we aim to provide investors with insights to make informed decisions.

Accurate stock price predictions have significant implications, allowing investors to identify optimal entry and exit points, manage risks, and optimize portfolios. Financial institutions can also leverage these predictions to offer valuable insights to their clients.

In this report, we discuss the methodologies employed in developing our prediction model, including data collection, feature engineering, and model selection. We present and analyze the results obtained, comparing the performance of our model against established benchmarks.

.

II. Data Exploring

1. About the dataset

The dataset we used is the historical price information for Google's stock, covering the period from March 2013 to the present. The dataset consists of 2510 rows and 7 columns which are:

- Date: the date on which the stock price was recorded.
- Price: The price at which the financial security opens in the market when trading begins
- High: represents the highest price at which the stock was traded during the day.
- Low: The lowest price of a share of Google stock on that date.
- Volume: The total number of shares of Google stock that were traded on that particular trading day.
- Close: The final trading price.
- Close Adj: The adjusted closing price of a share of Google stock on that date.

2. Data preprocessing

Stock price data is a prime example of time series data, which is a sequence of data points collected over time intervals. In the context of stock prices, each data point represents the price of a stock at a specific point in time.

a. Drop unused fields

During our data preprocessing stage, we identified that the "Close" and "Adj Close" features in our dataset were identical. As a result, we decided to remove the "Adj Close" feature from our dataset to reduce noise and avoid potential issues with multicollinearity. This step ensures that our model focuses on relevant features and improves the accuracy of our results.

b. Missing values

The dataset is complete, as it does not contain any missing values, thus eliminating the need for further data imputation or handling procedures.

c. Data Transformation

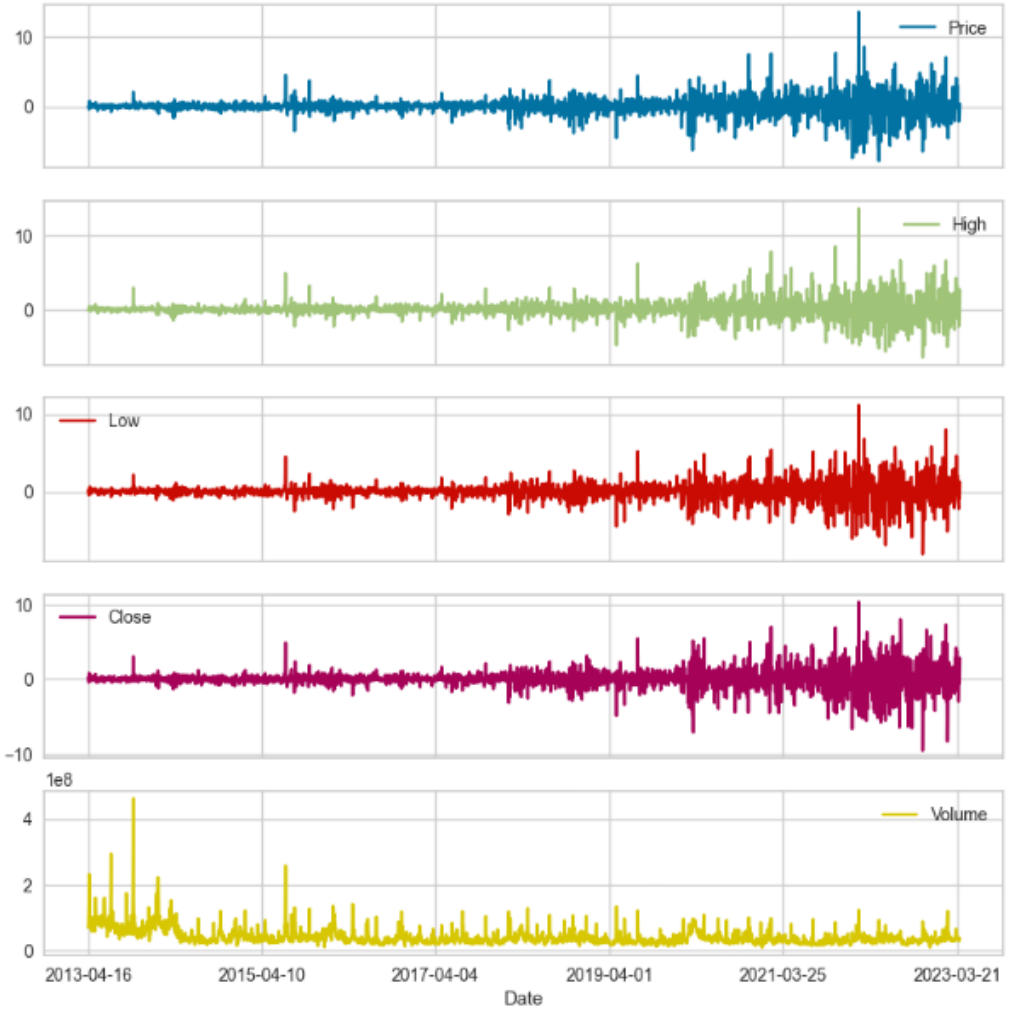
Stock prediction is a crucial task in finance and investment, and it involves analyzing time-series data. However, our analysis reveals that the data we are working with is non-stationary, meaning it exhibits trends, seasonality, and other patterns that change over time. This can make it challenging to identify underlying patterns in the data and predict future trends accurately.

To address this issue, we need to convert our non-stationary data into stationary data, which is more predictable and easier to model using machine learning techniques. One way to achieve this is by removing the trend and seasonality components through differencing or decomposition techniques.

We have conducted an Augmented Dickey-Fuller test on our data and found that four features - Price, High, Low, and Close - are non-stationary. Therefore, we have applied the differencing method to convert our data into stationary data.

$$X_s[i] = X_n[i+1] - X_n[i]$$

The resulting graph provides a visual representation of our converted data.



d. Feature selection

The ‘Close’ column serves as the target feature in our analysis, while the remaining columns are utilized to predict this target.

To analyze the correlation between variables, we construct a correlation matrix and visually represent it through a heatmap. This matrix provides insights into the strength and direction of the relationships between the variables.

From the above correlation heat map, it is observed that several column pairs with notably high correlation values variables, indicating a positive correlation. While high correlation between features can sometimes pose challenges, it is important to consider the context and relevance of these features.

In our case, we have a small number of features, including the target feature, which provides a limited feature space for predicting the output. Despite the high correlation between some feature pairs, these features also exhibit a high correlation with the dependent feature(the target feature). This suggests that they may contain valuable information and have a strong relationship with the target variable.

Given the limited feature set and the high correlation between these features and the target feature, it may be beneficial to include all of the features in the prediction model.

e. Feature Scaling

To ensure consistency and comparability in our data analysis, we employed standardization to scale the variables.

Standardization is the process of centering the variable at 0 (zero mean) and standardizing the variance to 1 (unit variance), and it is suitable for variables with a Gaussian distribution. After the standard scaling, the standard deviation will also be 1 (unit standard deviation). Remember that standard deviation = sqrt(variance), where sqrt is the square root.
To standardize features, we subtract the mean from each observation and then divide the result by the standard deviation:

$$X_{scaled} = \frac{X - u}{s}$$

The result of this transformation is called the z-score which indicates how many standard deviations a given observation deviates from the mean. Hence, standardization is also called z-score normalization.

By applying standardization to the Price, Open, Low, High and Close features, we ensured that these variables were on a comparable scale, enabling fair and meaningful comparisons and analysis throughout our study.

III. Methodology

1. Metrics

Choosing the right hyperparameters for a machine learning model is crucial because they can have a significant impact on the model's performance. Therefore, we need to use loss function to evaluate the performance after each model modification as well as evaluation between models. In this project, we use two evaluation methods: MSE and R-Squared.

Mean Squared Error (MSE) loss is calculated by squaring the difference between true value y and the predicted value $\hat{y}$. We take these new numbers (square them), add all of that together to get a final value, finally divide this number by y again.

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2$$

R-squared (R^2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable in a regression model.

$$\begin{aligned} R^2 &= 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}}, \\ &= 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}. \end{aligned}$$

In this machine learning report, we utilized the R- squared metric to assess the precision of our models. Subsequently, we compared the performance of the four models that we developed based on their R-squared scores.

2. Methods

Time Series forecasting involves employing a model to predict future values based on previously observed data. This can be achieved through the application of time series models. Alternatively, we can approach time series forecasting as a supervised learning problem, enabling the utilization of various machine learning models to predict the target variable. In the context of this specific problem, we can reframe it as a regression problem, where the objective is to predict a real value.

a. Linear Regression

Linear regression is widely used in practice and adapts naturally to even complex forecasting tasks. It is a simple yet effective modeling technique that assumes a linear relationship between a dependent variable and one or more independent variables. It aims to find a linear relationship that describes the data and enables prediction or inference based on the learned parameters.

Linear regression:

$$y = \alpha + \beta x$$

The simplest regression technique is called linear regression - when a single descriptive variable is being used and the advanced one called multiple regression - when more than one descriptive variable is used.

In the context of the problem we're working with, the technique we used is multiple linear regression. The technique predicts the future value of variable $\hat{Y}$ based on a vector of inputs $X^{T} = (X1, X2, \ldots\ldots, Xp)$, we predict the output Y via the model:

$$\hat{Y} = \widehat{\beta 0} + \sum_{j=1}^{p} Xj\,\widehat{\beta j}$$

The term $\alpha\ (or\ \widehat{\beta 0})$ is the intercept, also known as the bias in machine learning.
The term βj is called the weight or the parameter (regression coefficient) associated with the j-th independent variable.

If we include $\widehat{\beta 0}$ in the vector of coefficient $\hat{\beta}$ and then write the linear model in vector form:
$\hat{Y} = X^{T}\,\hat{\beta}$

During the training process, the regression algorithm learns values for the parameters set and the bias that best fit the target. To fit the linear model to a set of training data, there are many different methods. One commonly used approach is Ordinary Least Squares, which guides the learning process by minimizing the sum of squared differences between the predicted values and the actual values of the target variable.

Ridge Regression and Lasso Regression

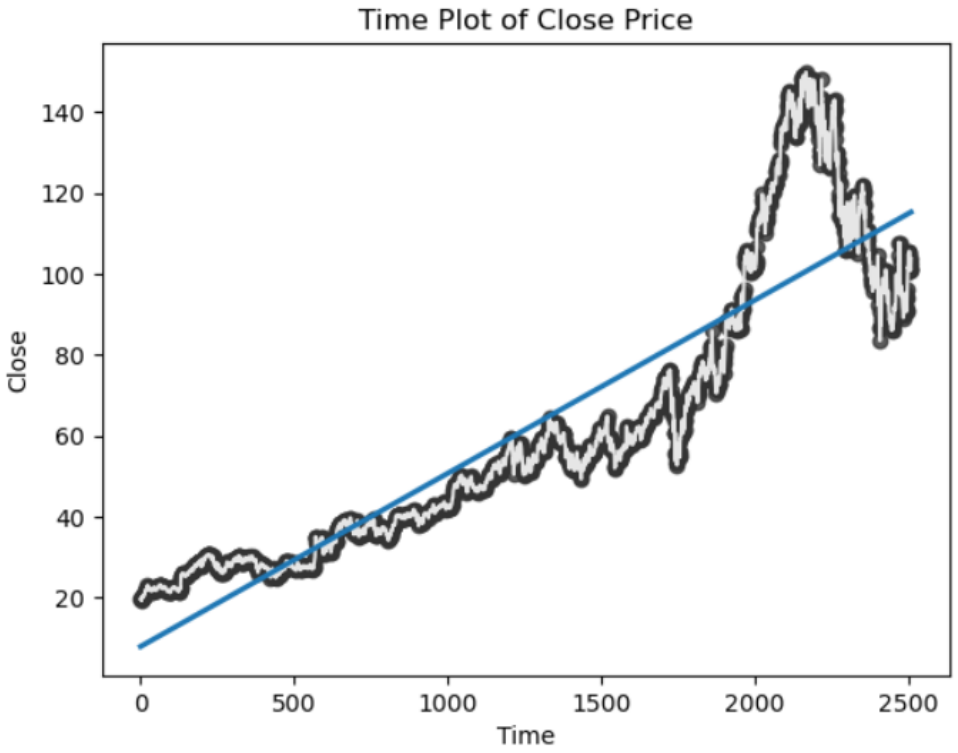
To avoid overfitting in Linear Regression model, there are two common regularization techniques: Ridge and Lasso Regression

Lasso Regression (L1 regularization): adds the “absolute value of magnitude” of the coefficient as a penalty term to the loss function.

$$Lasso\ Cost\ Function\ = \frac{1}{n} \sum_{i=1}^{n} n\ (yi\ - \widehat{yi})^2 + \alpha \sum_{j=1}^{p} |\beta j|$$

Ridge regression(L2 Regularization): adds the “squared magnitude” of the coefficient as the penalty term to the loss function.

$$Ridge\ Cost\ Function\ = \frac{1}{n} \sum_{i=1}^{n} n\ (yi\ - \widehat{yi})^2 + \alpha \sum_{j=1}^{p} \beta j^2$$



b. Random Forest Regressor (RF)

Decision Trees are a popular and intuitive method for solving regression problems, as they are quick, simple, and easy to interpret. However, there are certain limitations to this method. Decision Trees use a top-down greedy algorithm called ID3 to create nodes by selecting the best feature at each stage, which can lead to overfitting if the tree-building process is not carefully controlled.

To address this issue, Random Forest, a regression system that uses several decision trees, can be used. Random Forest creates a stochastic forest of trees using bagging and features randomness, with the aim of producing a prediction that is more accurate than that of any individual tree. Each tree in the forest only uses a subset of features and training points, which can help to reduce both bias and variance.

In our machine learning model, we use the variance reduction in each node of the decision trees as a measure of impurity. To calculate the variance of a tree node, we use the formula

$$Var = \frac{1}{n}\sum(y_i - \bar{y})^2$$

To determine the variance reduction, we calculate the weighted sum of the variance of the parent node and the variance of the child nodes.

$$VarRed = Var(parent) - Var(child_i)$$

The random forest algorithm then takes the average prediction result from the individual decision trees to predict the stock price.

In RF, we need to find some hyperparameters that optimize the model:

- N_estimators: The number of decision trees being built in the forest.
- Criterion: The function that is used to measure the quality of splits in a decision tree. In case of regression, MAE or MSE can be used.
- Max_depth: The maximum levels allowed in a decision tree.
- Min_samples_split: Decides the minimum number of samples required to split an internal node.
- Min_sample_leaf: The minimum number of data point requirements in a node of decision tree.
- Bootstrap: method for sampling data points (with or without replacement)

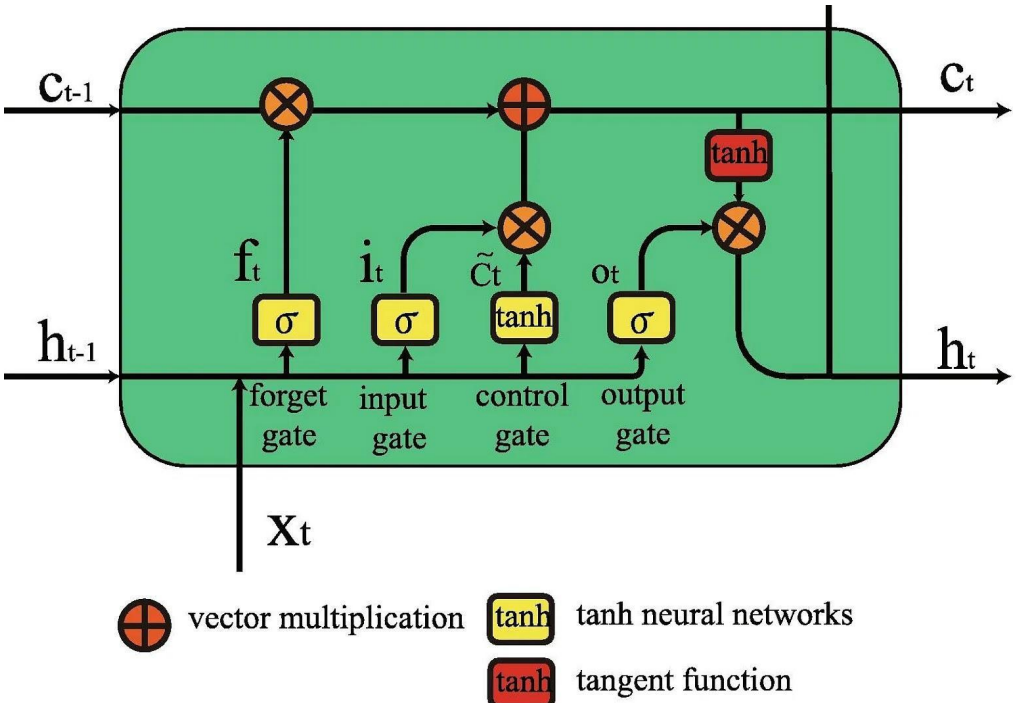
c. Long Short Term Memory (LSTM)

LSTM or Long Short-Term Memory is an improvement over traditional RNN or Recurrent Neural Network in the sense that it can effectively “remember” a long sequence of events in the past. Just like humans can derive information from the previous context and can chart his future actions, RNN and LSTM tend to imitate the same. The difference between RNN and LSTM is that RNN is used in places where the retention of memory is short, whereas

LSTM is capable of connecting events that happened way earlier and the events that followed them.

Hence, LSTM is one of the best choices when it comes to analyzing and predicting temporal dependent phenomena which span over a long period of time or multiple instances in the past.

A standard LSTM cell comprises three gates: the input, output, and forget gate. These gates learn their weights and determine how much of the current data sample should be remembered and how much of the past learned content should be forgotten. This simple structure is an improvement over the previous and similar RNN model.



As seen in the equations below, i, f, and o represent the three gates: input, forget, and output. C is the cell state that preserves the learned data, which is given as output h. All of this is computed for each timestamp t, considering the learned data from timestamp (t-1).

$$
\begin{aligned}
i_t &= \sigma(x_t U^t + h_{t-1} W^t) \\
f_t &= \sigma(x_t U^f + h_{t-1} W^f) \\
o_t &= \sigma(x_t U^f + h_{t-1} W^f) \\
\tilde{C}_t &= tanh(x_t U^g + h_{t-1} W^g) \\
C_t &= \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t \\
h_t &= tanh(C_t) * o_t
\end{aligned}
$$

The forget gate decides what information and how much of it can be erased from the current cell state, while the input gate decides what will be added to the current cell

state. The output gate, used in the final equation, controls the magnitude of output computed by the first two gates.

So, as opposed to standard feed-forward neural nets, LSTMs have the potential to remember or erase portions of the past data windows actively. Its feature of reading and training on windows (or timesteps) of data makes its training unique.

d. K-Nearest Neighbor for Regression

K-nearest neighbor technique is a machine learning algorithm that is considered as a simple yet effective machine learning algorithm for classification and regression tasks to implement. The stock prediction problem can be mapped into a similarity based regression. It also does not make any assumptions about the underlying data distribution. The historical stock data and the test data is mapped into a set of vectors. Each vector represents the N dimension for each stock features.Then, a similarity metric such as Euclidean distance is computed to make a decision. In this section, a description of kNN is provided. kNN is considered a lazy learning that does not build a model or function previously, but yields the closest k records of the training data set that have the highest similarity to the test (i.e. query record).Then, a majority vote is performed among the selected k records to take average value and then assigned as the prediction point z of the test data.

The prediction of stock market closing price is computed using kNN as follows:

1. Determine the number of nearest neighbors,|NB(z)| = k.
2. Compute the distance between the training samples and the new instance z. The similarity measure that we use is Euclidean distance:

$$d(x,z) = \sqrt{\sum_{i=1}^{n} (x_i - z_i)^2}$$

3. Sort all training records according to the distance values.
4. Compute the average of the target values among the k neighbors, and assign it as a prediction value of the new instance z.

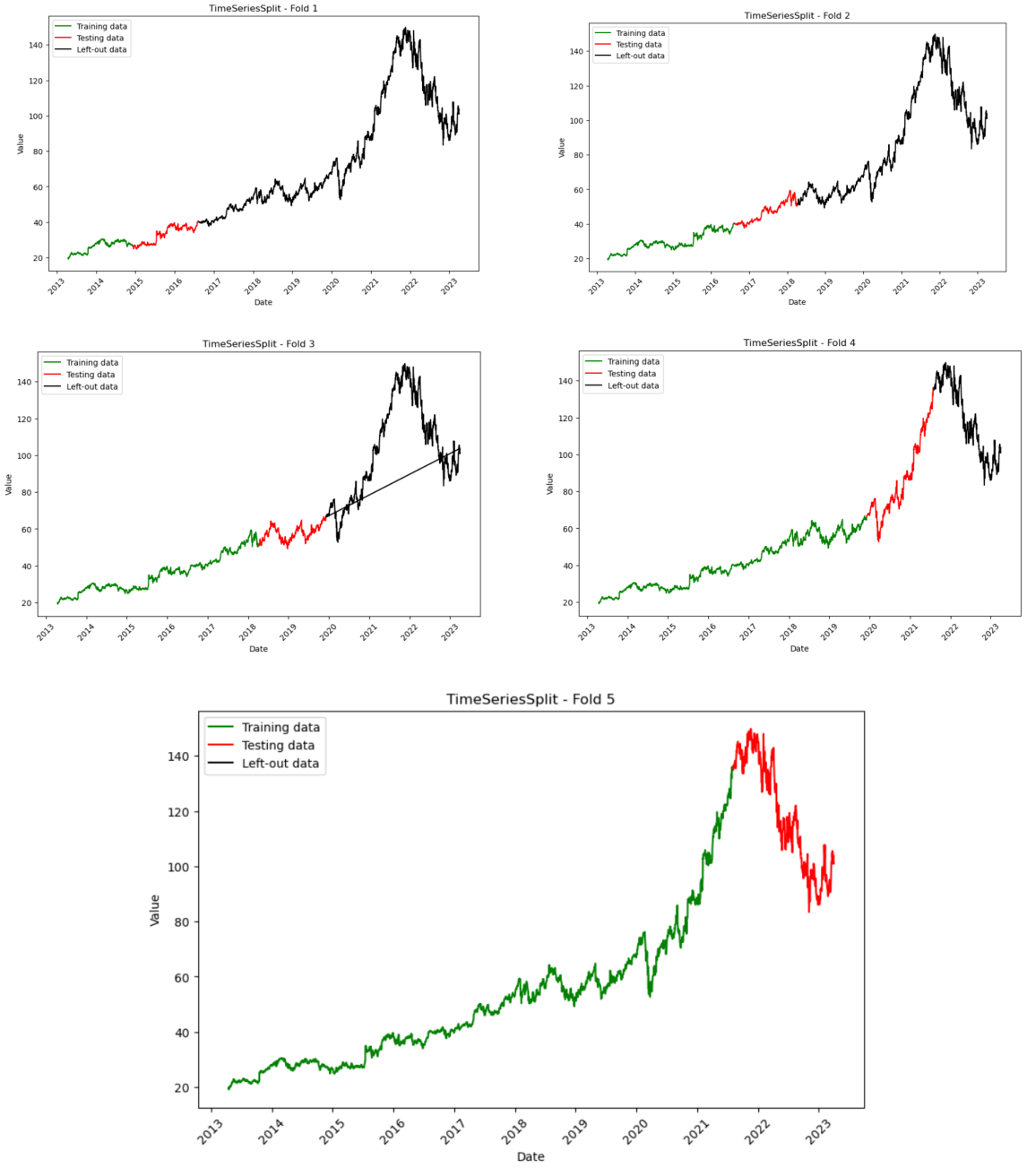
$$y_z = \frac{1}{k} \sum_{x \in NB(z)} y_x$$

Model selection.

To ensure optimal performance of each model, we employed the Grid Search technique with a 5-fold cross-validation strategy for hyperparameter tuning.

- **Grid search**: This method exhaustively explores various combinations of hyperparameters and selects the best score based on predefined evaluation metrics.
- **5 – fold Time Series Split**: Given that our data follows a time series structure, we used the TimeSeriesSplit method - which can be viewed as a variation of cross-validation specifically designed for time series data. The time series dataset is

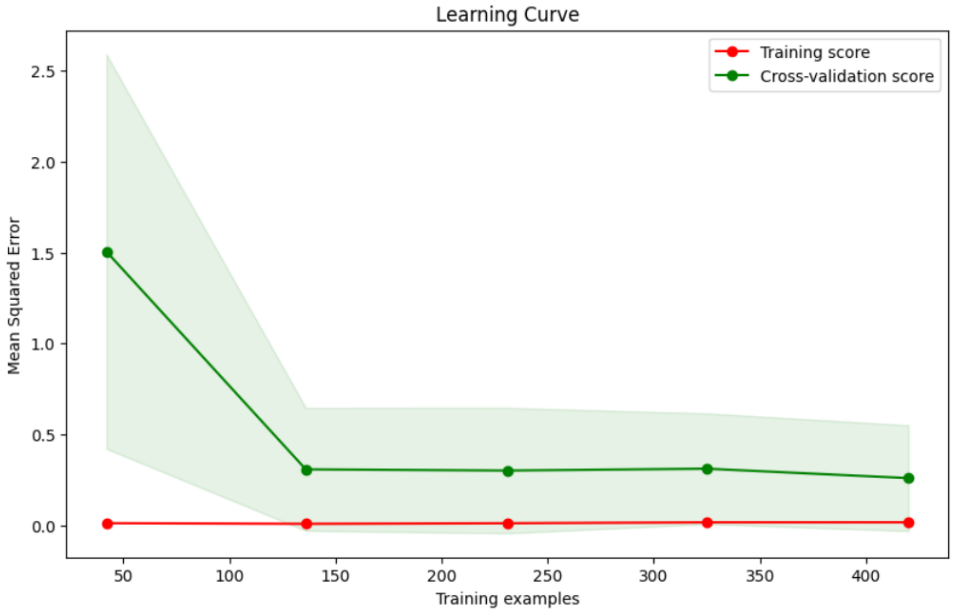
divided into 5 non-overlapping subsets of equal sizes. During each run of fold, one subset is reserved for testing while the remaining four subsets are used for training. This process is repeated five times, with each subset serving as the testing set once. Finally, the average score, computed based on the given evaluation metrics, is derived from the five runs, providing a robust estimation of model performance.

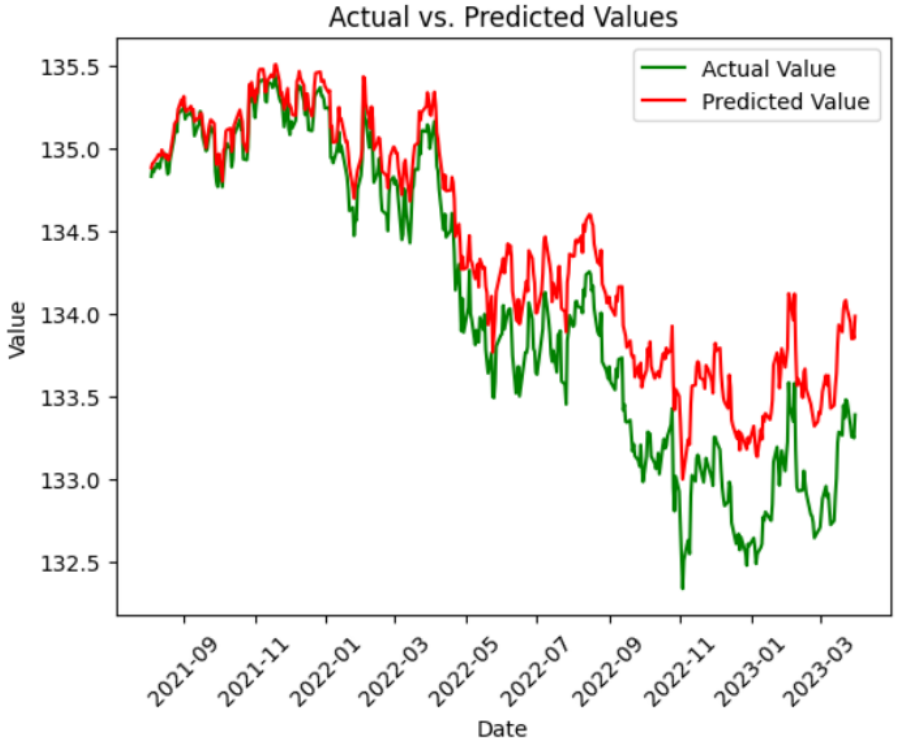


IV. Results

a. Linear regression

After applying L2 regularization and utilizing GridSearch to find the best parameters, I obtained results that include a learning curve graph. Upon analyzing the graph, it is evident that the model is performing well, as there are no indications of overfitting. This indicates that the regularization technique has effectively controlled the complexity of the model, preventing it from excessively fitting the training data.





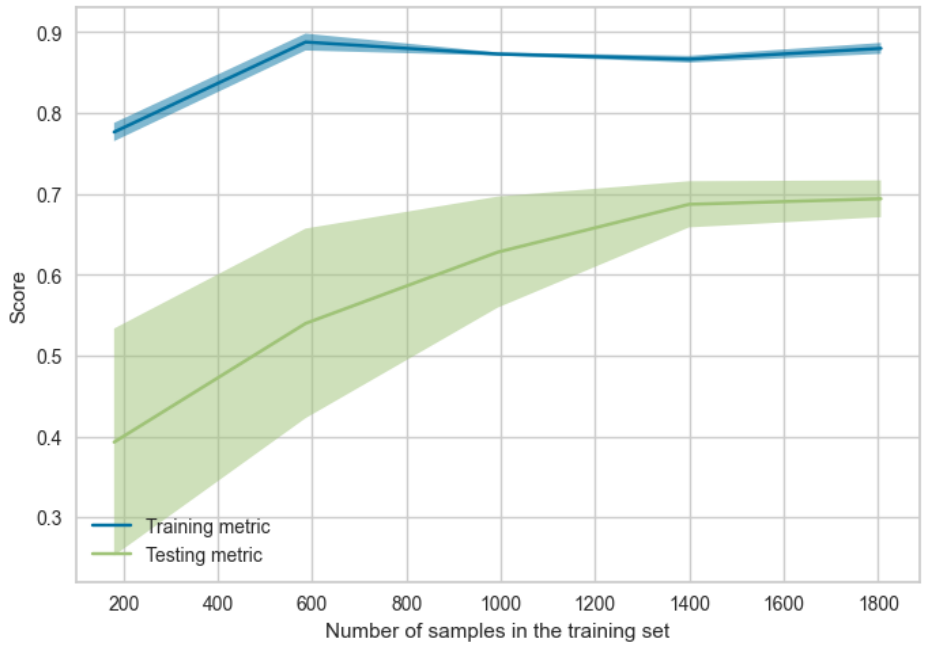
The comparison graph between the actual price and the predicted price on the data from 2021-09 to 2023-03

b. Random forest

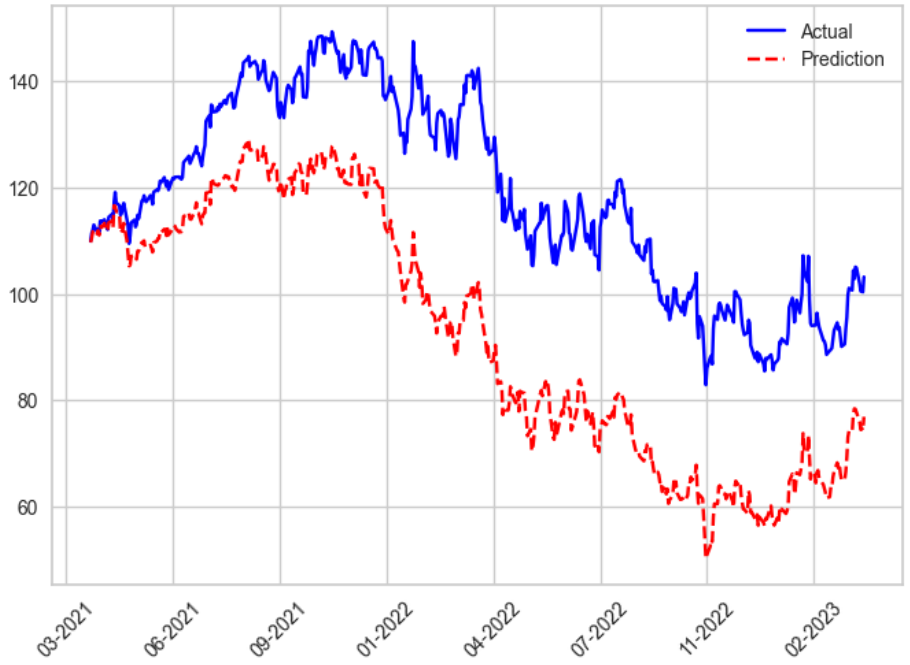
We utilized the RandomizedSearchCV function from Scikit-learn library to optimize our model's hyperparameters for our dataset. After tuning the hyperparameters, we obtained the following optimal values:

- n_estimators: 236
- min_samples_split: 2
- min_samples_leaf: 2
- max_depth: 10
- bootstrap: True

We then trained our model with these settings and plotted the learning curve to assess its performance.Our results showed that the model's accuracy gradually increased with additional training data until stabilizing at approximately 70%.

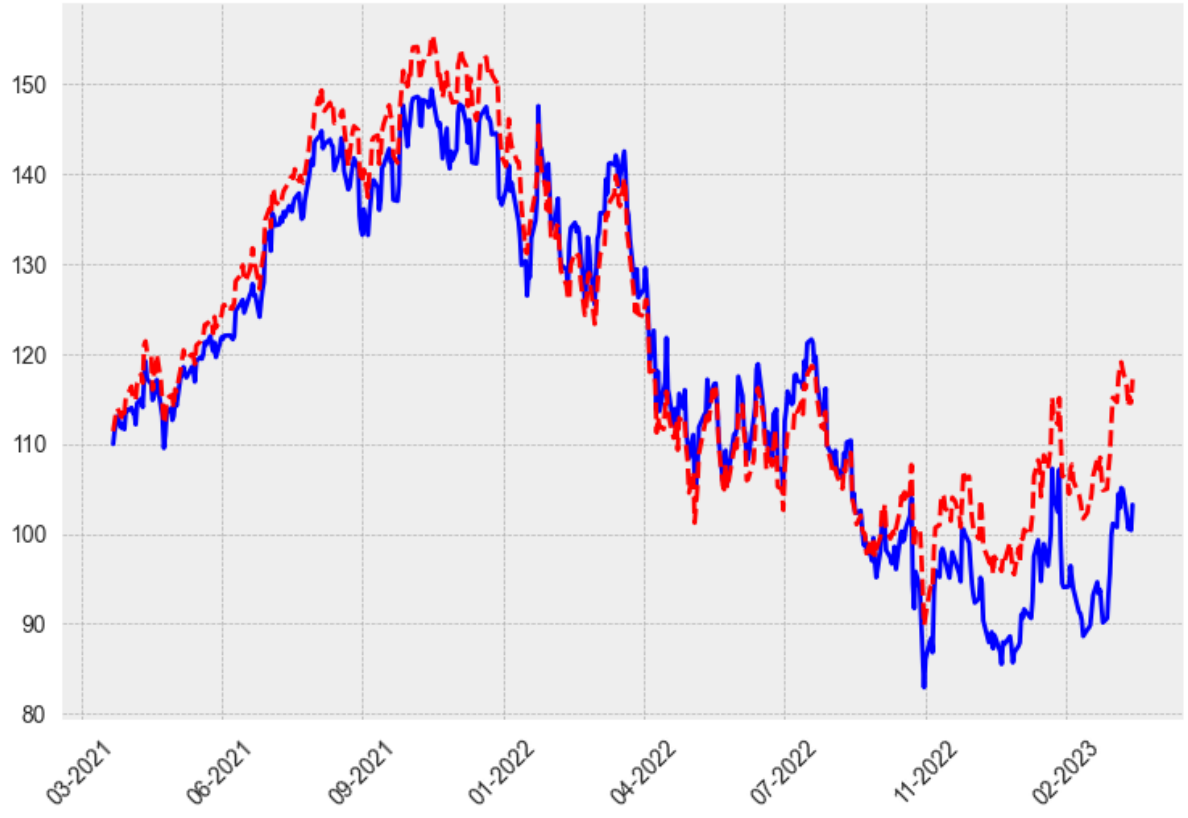


The graph below illustrates the difference between the predicted close values and the actual close values in our testing dataset:



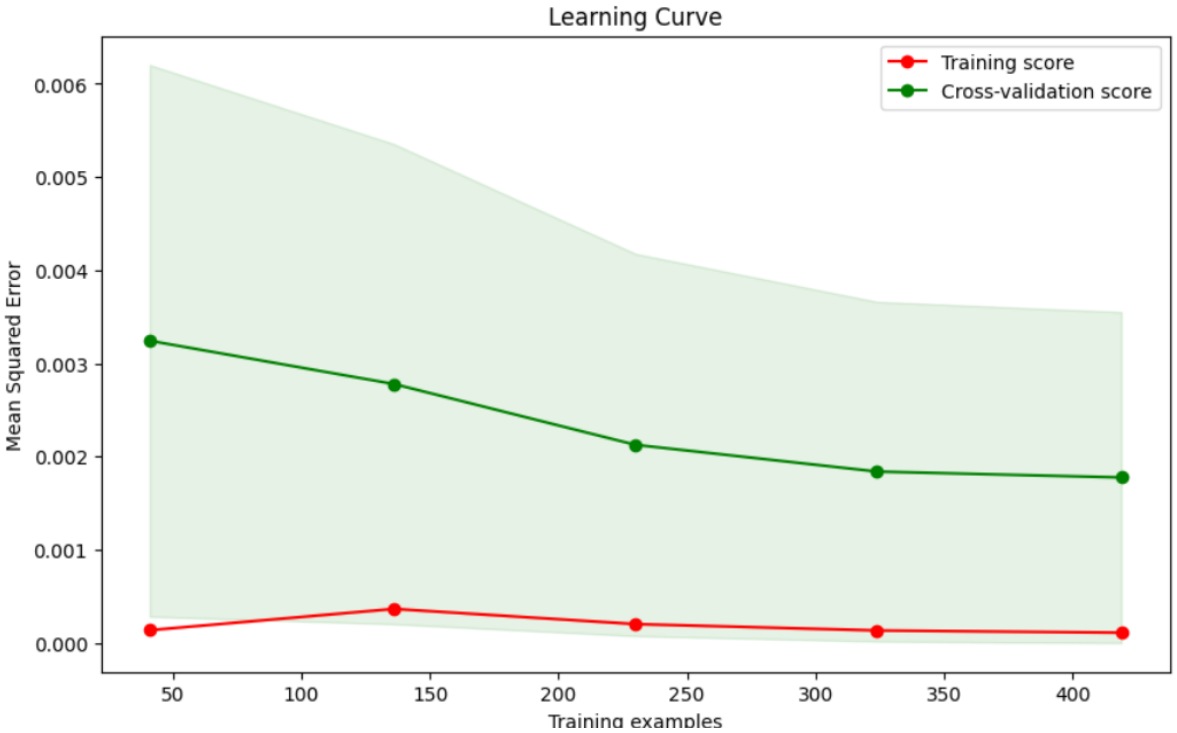
Using the R-Squared score, we obtained that the accuracy of the RF model is 71.79%.

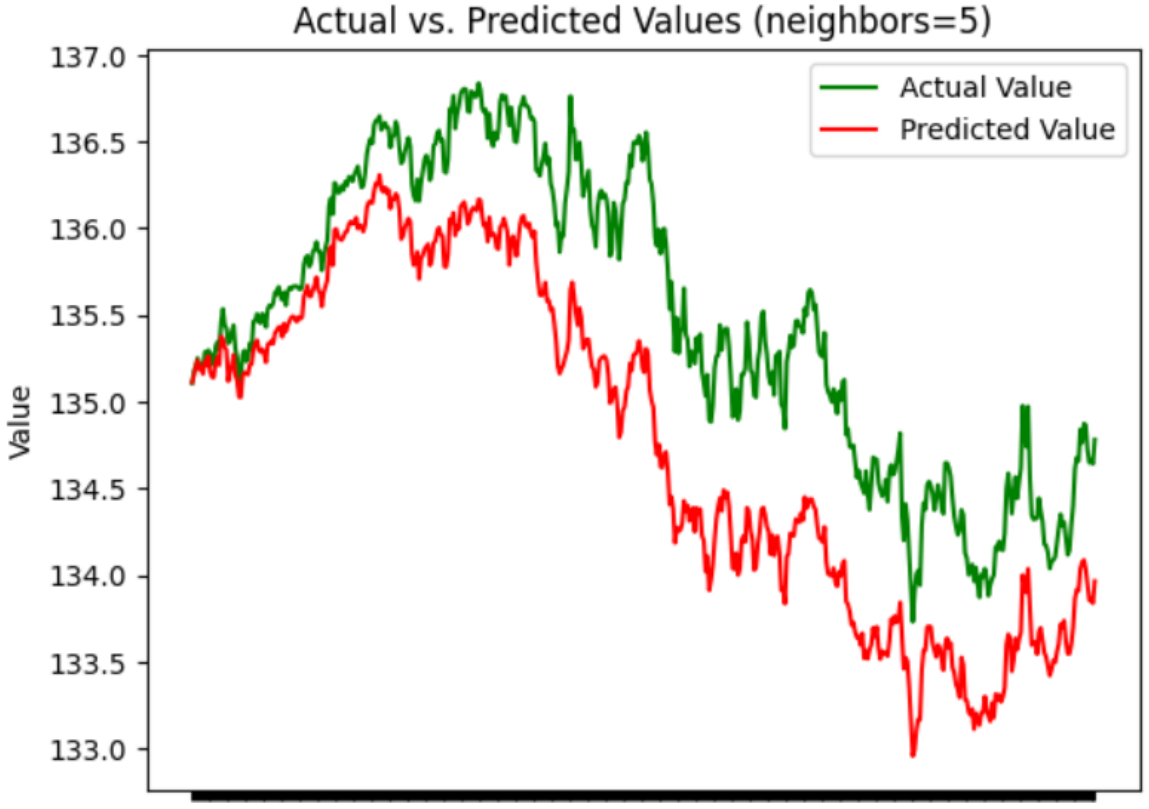
c. LSTM



Using the R-Squared score, we obtained that the accuracy of the LSTM model is 70.86%.

d. KNN


Learning Curve
Training score
Cross-validation score
Mean Squared Error
0.006
0.005
0.004
0.003
0.002
0.001
0.000
50
100
150
200
250
300
350
400
Training examples


Actual vs. Predicted Values (neighbors=5)
Actual Value
Predicted Value
Value
137.0
136.5
136.0
135.5
135.0
134.5
134.0
133.5
133.0

Comparative analysis

To compare the performance of the four models developed in this project, we calculated two key evaluation metrics: MSE and R2 score.

	Training set		Testing set	
Models	MSE	R2 Score	MSE	R2 Score
Linear Regression	0.001	0.7316	0.003	0.7787
Random Forest	0.099	0.8763	1.6277	0.7155
LSTM	0.1311	0.7732	0.148	0.7179
KNN	0.0007	0.7924	0.003	0.727

V. Conclusion and future work

From the obtained results, it is evident that all models perform well in predicting the target variable, but based on the R2 score (which serves as an accuracy metric) on the testing set, Linear Regression appears to work the best. However, it is important to note the context of the problem being addressed. In reality, predicting the stock price solely based on the information available during the day is not a realistic approach and may not be practically useful.

This project serves more as an experimental endeavor rather than a practical forecasting tool for long-term or future predictions. The limitations of the available information and the dynamic nature of the stock market make it challenging to accurately forecast prices over extended periods.

Looking ahead, the goal is to advance the model and refine its capabilities to make it more useful for investors and financial purposes. This involves incorporating additional relevant data, exploring more sophisticated algorithms, and considering other factors that influence stock prices. By continuing to enhance the model, the aim is to transform this project into a practical and valuable tool that can aid investors in making informed decisions.

VI. References

https://www.kaggle.com/code/ryanholbrook/linear-regression-with-time-series

https://machinelearningmastery.com/time-series-forecasting-supervised-learning/

https://towardsdatascience.com/exploring-the-lstm-neural-network-model-for-time-series-8b7685aa8cf

https://medium.com/@maryamuzakariya/project-predict-stock-prices-using-random-forest-regression-model-in-python-fbe4edf01664

https://www.analyticsvidhya.com/blog/2021/06/random-forest-for-time-series-forecasting/

https://analyticsindiamag.com/how-to-make-a-time-series-stationary/

https://www.jmlr.org/papers/volume14/guyader13a/guyader13a.pdf