VISUALISING AND FORECASTING STOCKS

Group Members:

Pawan Kumar
Lovely Professional University
12100975

Kovidsai Vemuri
Lovely Professional University
12114491

Contents

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty, **Prof. Arnab Chakraborty** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Pawan Kumar

Kovidsai Vemuri

Project Objective

In this project we have taken 'Stock Market' Dataset from Yahoo Finance. In this dataset, the target attribute is the stock price. So, in this project we need to do classification based on the attributes present in our dataset and predict what will be the stock price of the given stock for the next upcoming days.

Our objective in this project is to study the given dataset of 'Stock Market'. We might need to pre-process the given dataset if we need to. Then, we would train 4 models viz. 'Support Vector Regression Model', 'Linear Regression Classifier Model', 'Decision Tree classifier model' and 'Long Short Term Memory Model'. After training the aforementioned models, we will need to find out the score, classification report, plot the Receiver Operating Characteristic graph for each of the models trained. Our next step would be to use the trained models to predict the outcomes using the given test dataset and compare the outcome of each model. We would then choose the best model based on the accuracy score and classification report.

Our methodology for solving the problems in the given project is described below:

- Load the required dataset.
- Study the dataset.
- Describe the dataset.
- Visualise the dataset.
- Find out if the dataset needs to be pre-processed.

o It will be determined on the basis of whether the dataset has null values or outliers or any such discrepancy that might affect the output of the models to be trained.

- If the dataset is required to be pre-processed, take the necessary steps to pre-process the data.
- Find out the principal attributes for training.

- Split the given dataset for training and testing purpose.
- Fit the previously split train data in the aforementioned 4 models.
- Calculate the accuracy of the 4 models and find out the classification reports.
- Plot the necessary graphs.
- Use each trained model to predict the outcomes of the given test dataset.
- Choose the best model among the 4 trained models bases on the accuracy and classification reports.

Project Scope

The broad scope of ‘Visualising and Forecasting Stocks’ project is given below:

- The given dataset has attributes based on which the stock price of the given stock will be predicted.
- It is a useful project as the Classifier models can be used to quickly determine the stock price of the stock of large datasets.
- Various Financial institutions and traders can use these models and modify them according to their needs to predict stock market price. This will reduce the manual work and time spent on determining whether the the stock of the partical company will be open or close at that price or not.
- Customers who want to predict the stock market price can use these trained models to check whether the price of partical stock will go up or down. The stock will be opened or closed at which price. The trained models would be required to be implemented in a platform or interface easily accessible as well as with an easy GUI.
- The dataset given to us is a dataset taken from Yahoo Finance. So, the results might have some mismatch with the real-world applications. But that can be avoided if the models are trained accordingly.

Data Description

Source of the data: Yahoo Finance. The given dataset is a shortened version of the original dataset in Yahoo Finance.

Data Description: The given train dataset has 1234 rows and 6 columns.

Columns	**Attribute Name**	**Type**	**Description**	**Target Attribute**
Open	Open	Categorical	The price of the stock at the beginning of the trading day.	Yes
High	High	Categorical	The highest price reached by the stock during the trading day.	No
Low	Low	Categorical	The lowest price reached by the stock during the trading day.	No
Close	Close	Categorical	The price of the stock at the end of the trading day.	No
Adj Close	Adj Close	Categorical	The closing price adjusted for factors such as dividends, stock splits, or other corporate actions.	No
Volume	Volume	Categorical	The number of shares traded during the trading day.	No

Table 1: Data Description

The following table shows the 10 number statistics of the given dataset:

Date	Open	High	Low	Close	Adj Close	Volume
2018-07-16	118.933334	119.833336	116.766663	118.599998	94.274284	8057787
2018-07-17	119.333336	122.116669	119.000000	119.566666	95.042686	10419621
2018-07-18	120.183334	121.849998	118.433334	119.683334	95.135422	10709484
2018-07-19	120.133331	121.349998	118.583336	119.266663	94.804199	5064312
2018-07-20	119.599998	120.933334	119.216667	119.933334	95.334145	5851905
2018-07-23	120.466667	121.650002	118.783333	119.599998	95.069183	9013209
2018-07-24	119.633331	123.099998	118.466667	121.933334	96.923927	13471506
2018-07-25	123.000000	123.516663	120.733330	121.550003	96.619232	8770071
2018-07-26	121.550003	124.083336	120.966667	123.333336	98.036774	25113513
2018-07-27	123.716667	126.599998	122.949997	126.166664	100.288963	10968603

Table 2: 10 number statistics of the given dataset

Data Pre-Processing

Now we will pre-process the data. The methodology followed is given below:

- Checking for null values.

  If null values are present, we will fill them or drop the row containing the null value based on the dataset.

- Checking for outliers.

  If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

Now we will pre-process the data. The methodology followed is given

below:

We searched for null values in our dataset

```
Null values:
Open          0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
```

• Checking for null values. o If null values are present, we will fill them or

drop the row containing the null value based on the dataset.

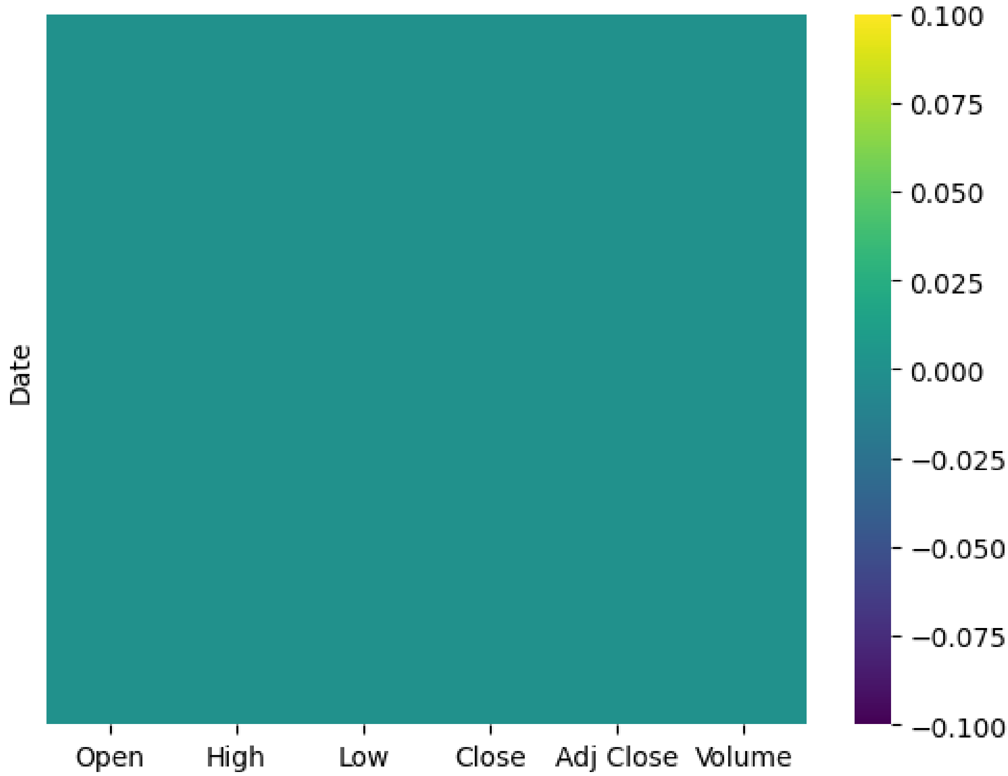
• Checking for outliers. o If outliers are present, they will either be removed

or replaced by following a suitable method depending on the dataset.

Or can be replaced using the module LabelEncoder from sklearn.

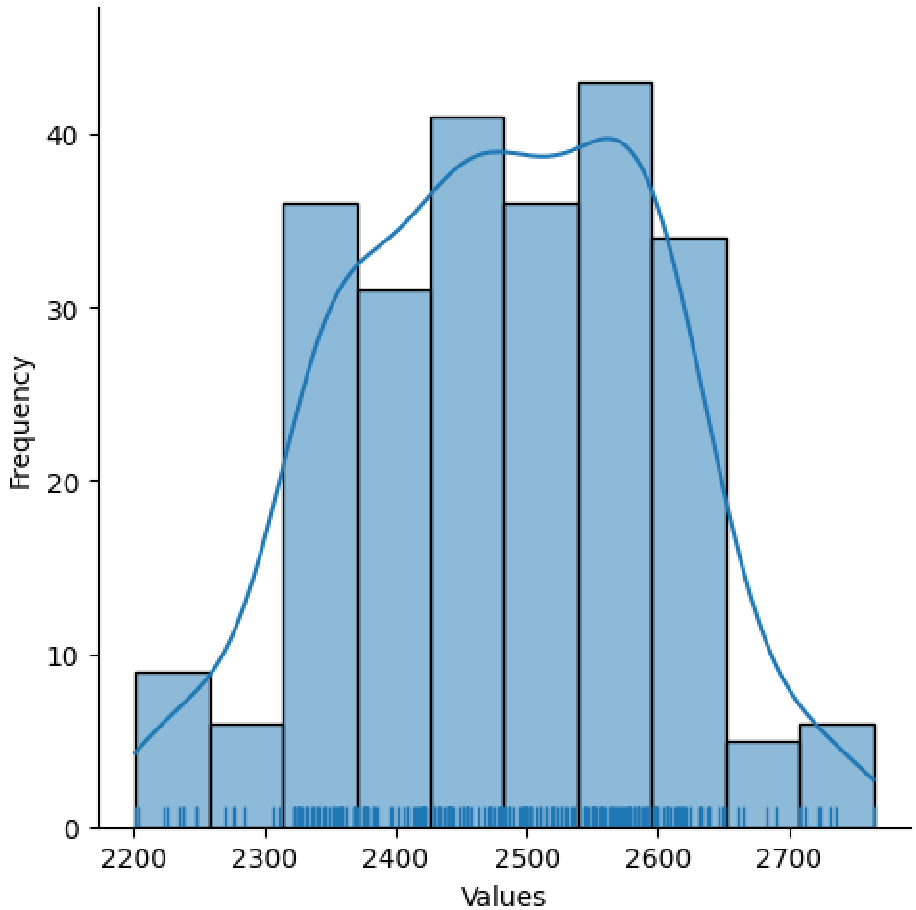
To visualise the null values, we made a heatmap plot using seaborn library

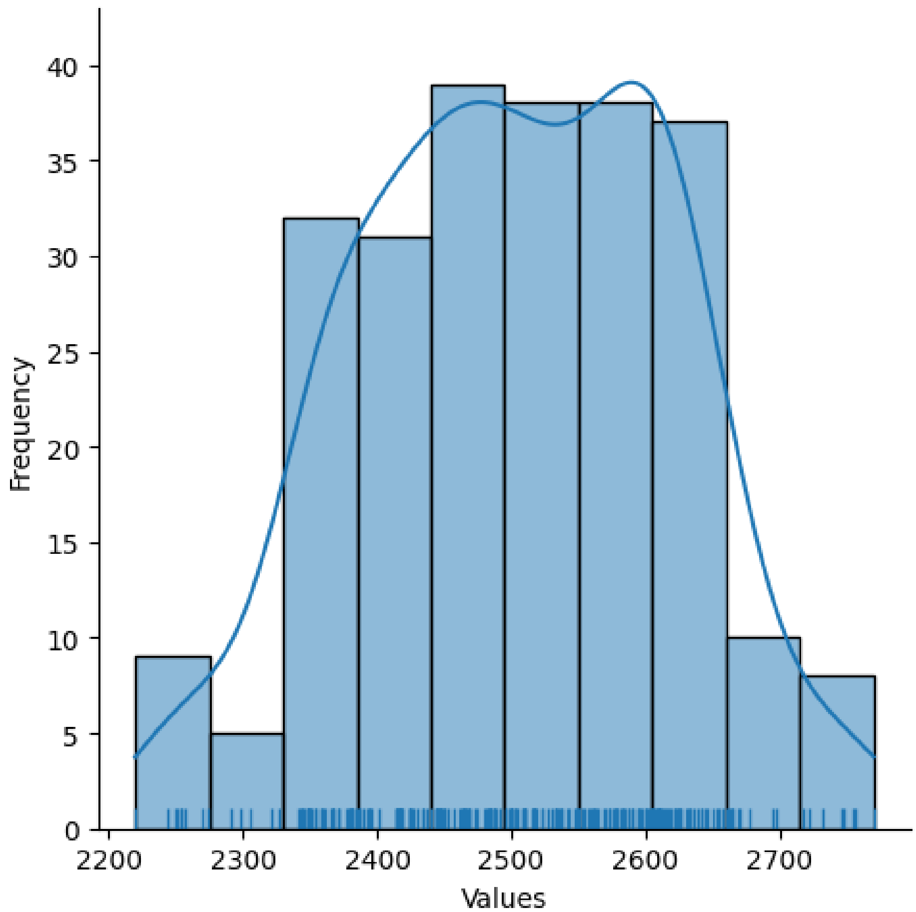
function heatmap. The heatmap plot is given below:



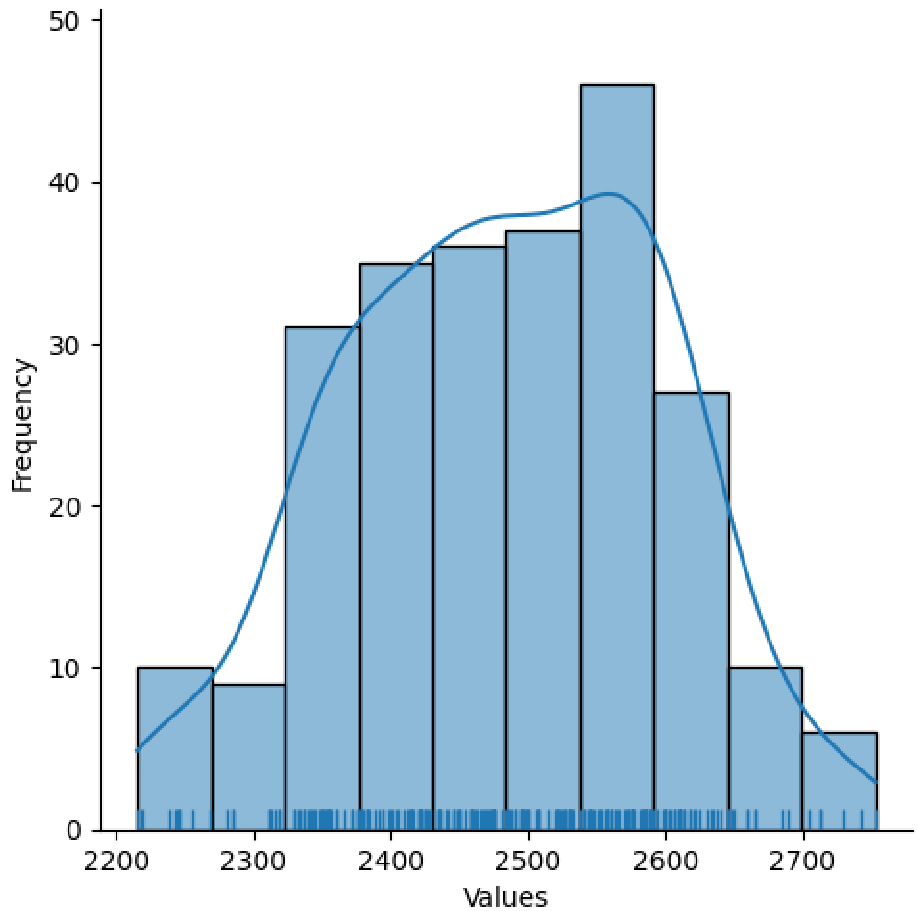
No Null Values in the Data Set.

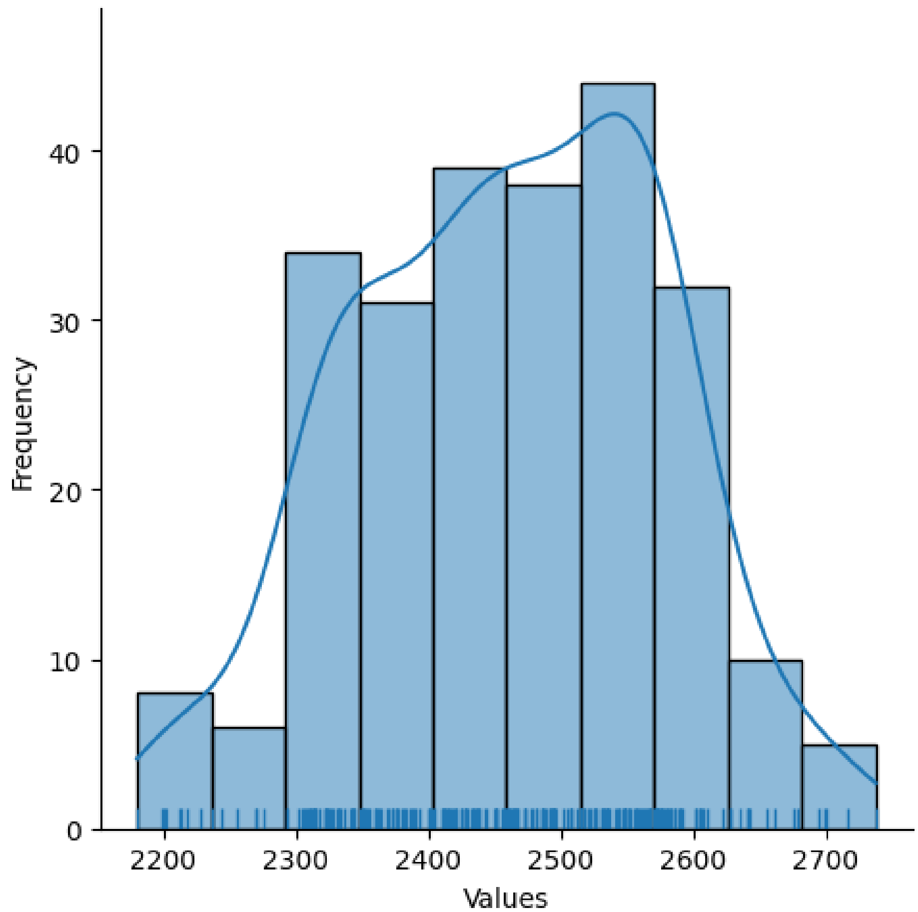
Checking for Outliers:





0
5
10
15
20
25
30
35
40
Frequency
2200
2300
2400
2500
2600
2700
Values


50
40
30
20
10
0
Frequency
2200
2300
2400
2500
2600
2700
Values




```
# Detect outliers using the IQR method
outliers = ((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 *
IQR))).any(axis=1)

# Print rows with outliers
print("Rows with outliers:")
print(data[outliers])
```

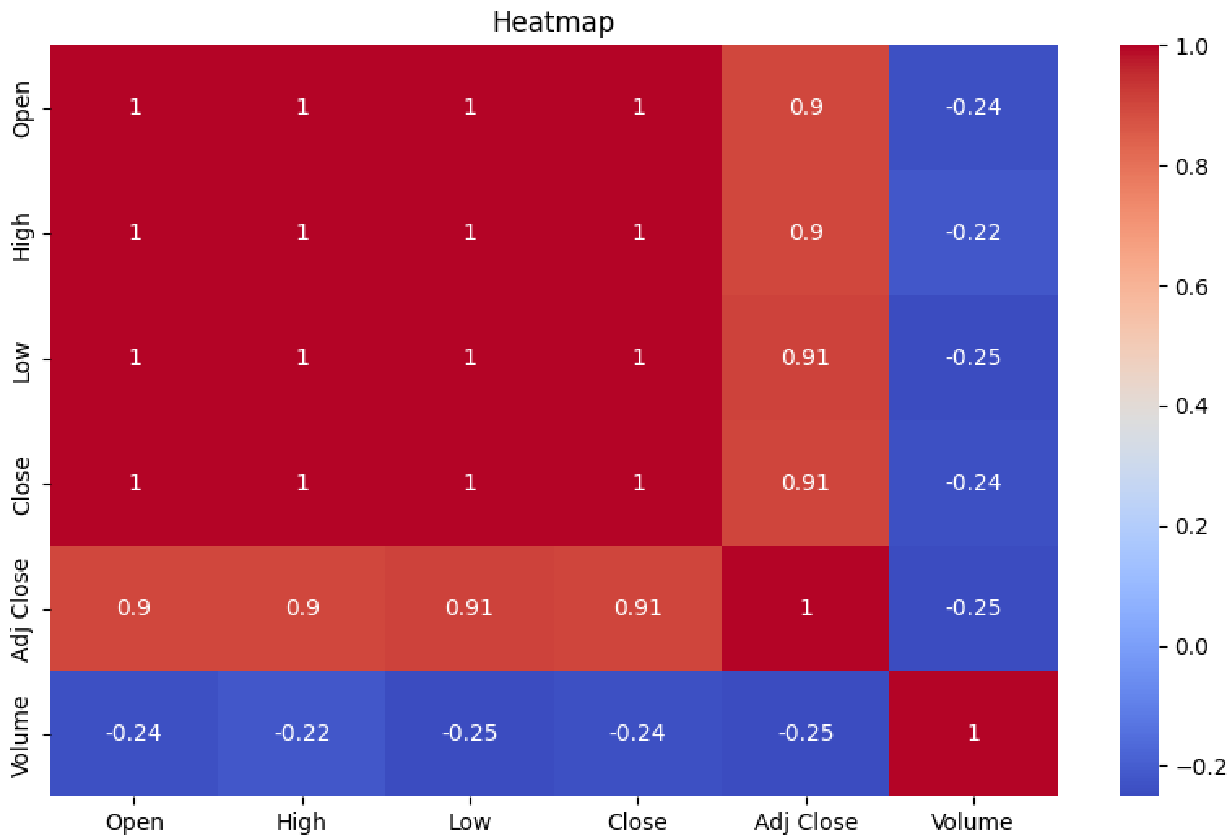
```
Rows with outliers:
             Open         High          Low        Close    Adj Close
Volume
6    2540.000000  2542.500000  2486.250000  2503.000000  2495.486816
11041036
93   2608.899902  2721.050049  2502.000000  2707.550049  2707.550049
14549929
95   2712.500000  2745.449951  2698.199951  2731.350098  2731.350098
12075137
136  2384.399902  2387.350098  2311.649902  2337.350098  2337.350098
11920991
141  2349.000000  2349.000000  2293.000000  2329.000000  2329.000000
11398850
149  2376.000000  2437.199951  2373.000000  2431.949951  2431.949951
15461902
170  2244.750000  2251.949951  2212.699951  2223.100098  2223.100098
15697554
179  2255.000000  2343.449951  2254.699951  2331.050049  2331.050049
13001005
218  2500.000000  2509.850098  2461.000000  2469.899902  2469.899902
12510304
230  2560.199951  2582.399902  2560.199951  2577.399902  2577.399902
11155180
245  2688.899902  2756.000000  2675.000000  2735.050049  2735.050049
15340262
```

Removing Rows With Outliers

```python
dataT = data[~outliers]

# Print the modified dataset without outliers
print("Modified dataset without outliers:")
print(dataT)
```

```
Modified dataset without outliers:
            Open         High          Low        Close    Adj Close   Volume
0    2404.000000  2439.699951  2404.000000  2420.449951  2413.184570  4974502
1    2427.300049  2434.000000  2373.000000  2377.550049  2370.413330  6564435
2    2388.000000  2433.949951  2376.949951  2397.149902  2389.954346  7831798
3    2415.000000  2415.000000  2383.100098  2401.800049  2394.590576  4431880
4    2421.000000  2425.000000  2392.300049  2422.250000  2414.979248  6996757
..           ...          ...          ...          ...          ...      ...
241  2625.000000  2625.000000  2573.250000  2588.750000  2588.750000  3720447
242  2609.000000  2609.000000  2575.800049  2584.500000  2584.500000  4729479
243  2576.050049  2644.449951  2576.050049  2638.750000  2638.750000  8822948
244  2635.000000  2664.949951  2628.000000  2633.600098  2633.600098  6172684
246  2752.899902  2770.000000  2737.600098  2764.699951  2764.699951  9250766

[236 rows x 6 columns]
```

Corelation Heatmap:



Correlation Table:

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.986928	0.986687	0.971127	0.971052	-0.272168
High	0.986928	1.000000	0.985403	0.990341	0.990213	-0.217832
Low	0.986687	0.985403	1.000000	0.986007	0.986029	-0.310188
Close	0.971127	0.990341	0.986007	1.000000	0.999811	-0.238590
Adj Close	0.971052	0.990213	0.986029	0.999811	1.000000	-0.240194
Volume	-0.272168	-0.217832	-0.310188	-0.238590	-0.240194	1.000000

Model Building

LSTM:

LSTM is an artificial recurrent neural network architecture. Artificial neural networks is a branch of machine learning where the analytical model is built using a layered architecture, where each layer transforms the input data it is given and outputs one or multiple output values. Each layer is comprised of a set number of nodes (called neurons). Each neuron takes some input and performs a linear transformation on it, before applying a nonlinear function on the transformed value and outputting the result. This procedure is performed on the input for each neuron in a layer, but with different weights for each neuron. Afterwards, the outputs may either be used for classification or regression, or sent along to another layer. The process of adjusting the weights of all neurons in such a way so that they give the desired output is the training phase of artificial neural networks.

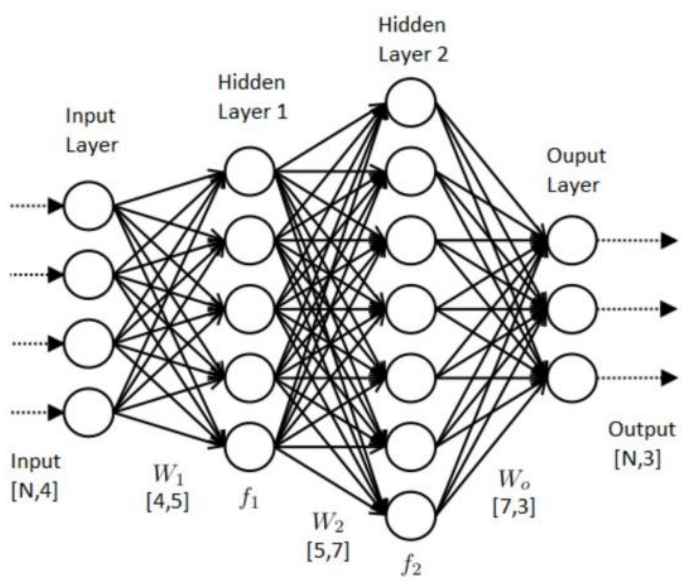


Fig: An artificial neural network with four input dimensions, two hidden layers and three output dimensions.

As a recurrent neural network (RNN), LSTMs are able to process both single data points (e.g. one set of four values as in figure 2.1) but also data points in a sequence such as videos or audio. An issue with a basic (a.k.a. "vanilla") RNN is that it is unable to learn patterns that transpire over a long sequence (i.e. over a long period of time), e.g. a pattern relating the beginning and end of a video. LSTMs, created by Hochreiter and Schmidhuber [3], solve this by employing a more complex architecture comprised of multiple smaller units that control what memory to forget and what to keep as training progresses, see figure 2.2. This improved architecture allows them to not only learn local patterns similar to vanilla RNNs, but also enables them to learn patterns that transpire over longer periods.

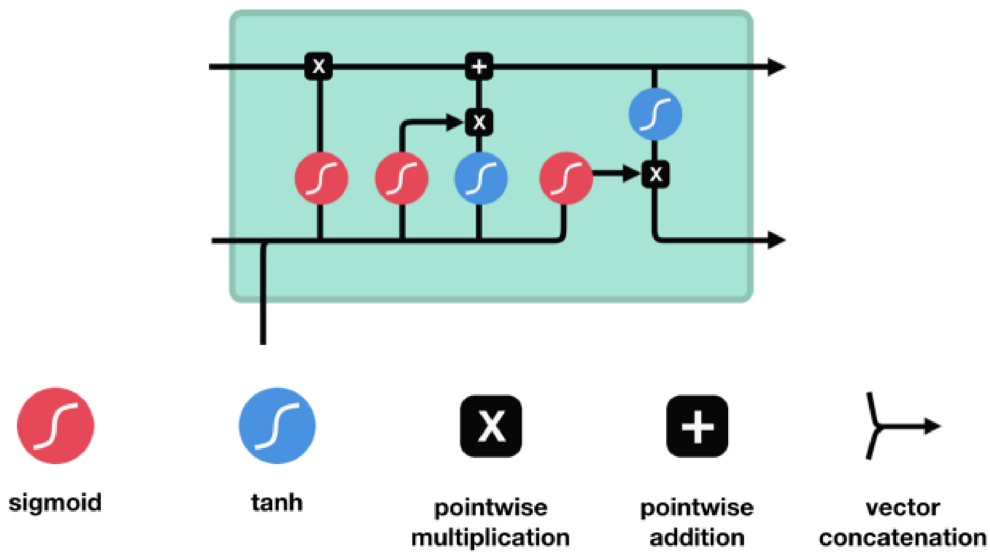


Fig: An LSTM and its units. Two outputs, one of which is the memory itself and the other the current output for the given input. The memory and output of the previous time step is also given as input to the next time step, along with a new data point.

SVR:

Support Vector Regression (SVR) is a type of machine learning algorithm used for regression analysis. The goal of SVR is to find a function that approximates the relationship between the input variables and a continuous target variable, while minimizing the prediction error.

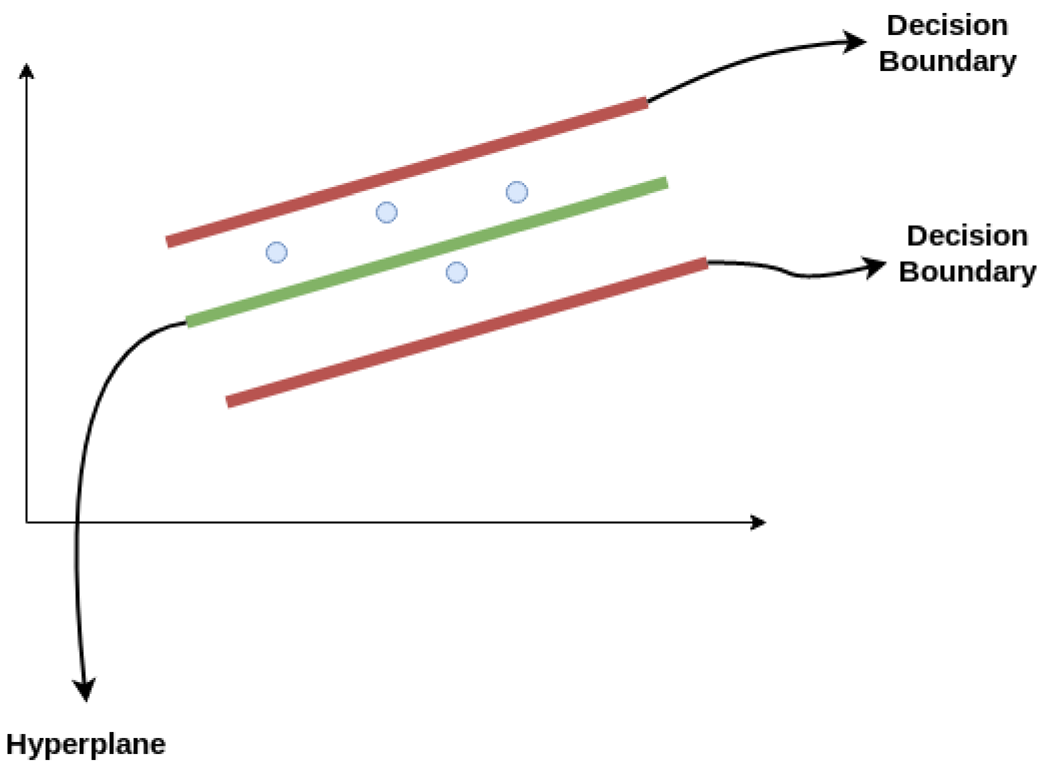
Unlike Support Vector Machines (SVMs) used for classification tasks, SVR seeks to find a hyperplane that best fits the data points in a continuous space. This is achieved by mapping the input variables to a high-dimensional feature space and finding the hyperplane that maximizes the margin (distance) between the hyperplane and the closest data points, while also minimizing the prediction error.

SVR can handle non-linear relationships between the input variables and the target variable by using a kernel function to map the data to a higher-dimensional space. This makes it a powerful tool for regression tasks where there may be complex relationships between the input variables and the target variable.

Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems. Let's spend a few minutes understanding the idea behind SVR.

The Idea Behind Support Vector Regression

The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. So let's now dive deep and understand how SVR works actually.



Consider these two red lines as the decision boundary and the green line as the hyperplane. **Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line.** Our best fit line is the hyperplane that has a maximum number of points.

The first thing that we'll understand is what is the decision boundary (the danger red line above!). Consider these lines as being at any distance, say 'a', from the hyperplane. So, these are the lines that we draw at distance '+a' and '-a' from the hyperplane. This 'a' in the text is basically referred to as epsilon.

Assuming that the equation of the hyperplane is as follows:

```
Y = wx+b (equation of hyperplane)
```

Then the equations of decision boundary become:

```
wx+b= +a
```

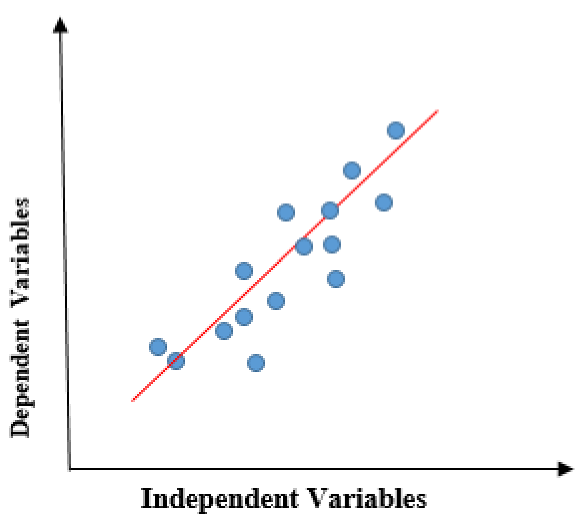
```
wx+b= -a
```

Thus, any hyperplane that satisfies our SVR should satisfy:

```
-a < Y- wx+b < +a
```

Linear Regression:

Linear regression is a quiet and simple statistical regression method used for predictive analysis and shows the relationship between the continuous variables. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), consequently called linear regression. *If there is a single input variable (x), such linear regression is called* ***simple linear regression****. And if there is more than one input variable, such linear regression is called* ***multiple linear regression.*** The linear regression model gives a sloped straight line describing the relationship within the variables.



The above graph presents the linear relationship between the dependent variable and independent variables. When the value of x (**independent variable**) increases, the value of y (**dependent variable**) is likewise increasing. The red line is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best.

To calculate best-fit line linear regression uses a traditional slope-intercept form.

$$y = mx+b \quad \Longrightarrow \quad y = a_0+a_1x$$

y= Dependent Variable.

x= Independent Variable.

a0= intercept of the line.

a1 = Linear regression coefficient.

Need of a Linear regression

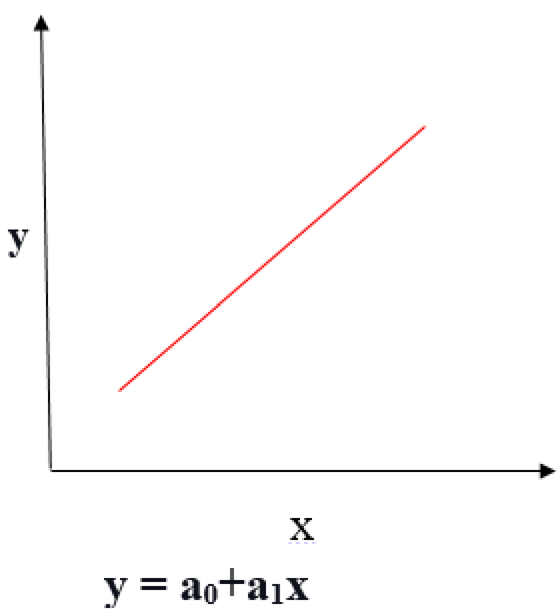
As mentioned above, Linear regression estimates the relationship between a dependent variable and an independent variable. Let’s understand this with an easy example:

Let’s say we want to estimate the salary of an employee based on year of experience. You have the recent company data, which indicates that the relationship between experience and salary. Here year of experience is an independent variable, and the salary of an employee is a dependent variable, as the salary of an employee is dependent on the experience of an employee. Using this insight, we can predict the future salary of the employee based on current & past information.

A regression line can be a Positive Linear Relationship or a Negative Linear Relationship.

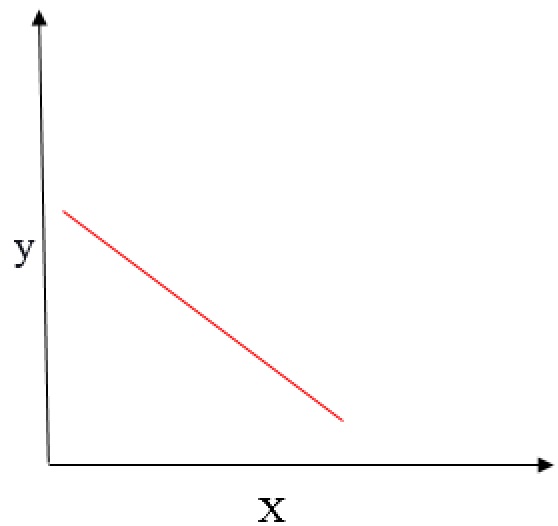
Positive Linear Relationship

If the dependent variable expands on the Y-axis and the independent variable progress on X-axis, then such a relationship is termed a Positive linear relationship.



Negative Linear Relationship

If the dependent variable decreases on the Y-axis and the independent variable increases on the X-axis, such a relationship is called a negative linear relationship.



$y = -a_0 + a_1 x$

The goal of the linear regression algorithm is to get the best values for a0 and a1 to find the best fit line. The best fit line should have the least error means the error between predicted values and actual values should be minimized.

Cost function

The cost function helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points.

Cost function optimizes the regression coefficients or weights and measures how a linear regression model is performing. The cost function is used to find the accuracy of the **mapping function** that maps the input variable to the output variable. This mapping function is also known as **the Hypothesis function**.

In Linear Regression, **Mean Squared Error (MSE)** cost function is used, which is the average of squared error that occurred between the predicted values and actual values.

By simple linear equation y=mx+b we can calculate MSE as:

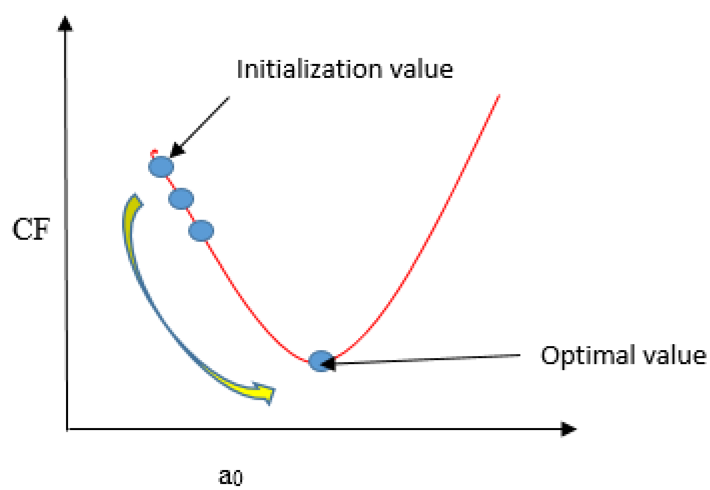
Let's y = actual values, y_i = predicted values

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

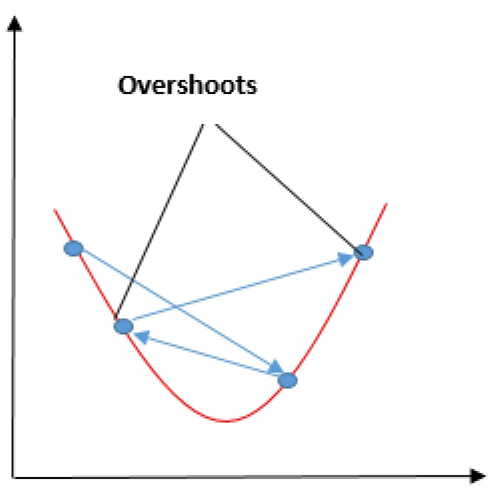
Using the MSE function, we will change the values of a0 and a1 such that the MSE value settles at the minima. Model parameters ***xi, b*** ***(a_0,a_1)*** can be manipulated to minimize the cost function. These parameters can be determined using the gradient descent method so that the cost function value is minimum.

Gradient descent

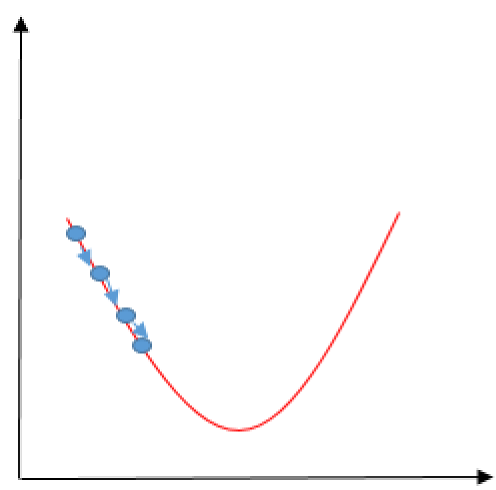
Gradient descent is a method of updating a0 and a1 to minimize the cost function (MSE). A regression model uses gradient descent to update the coefficients of the line (a0, a1 => xi, b) by reducing the cost function by a random selection of coefficient values and then iteratively update the values to reach the minimum cost function.



Imagine a pit in the shape of U. You are standing at the topmost point in the pit, and your objective is to reach the bottom of the pit. There is a treasure, and you can only take a discrete number of steps to reach the bottom. If you decide to take one footstep at a time, you would eventually get to the bottom of the pit but, this would take a longer time. If you choose to take longer steps each time, you may get to sooner but, there is a chance that you could overshoot the bottom of the pit and not near the bottom. In the gradient descent algorithm, the number of steps you take is the learning rate, and this decides how fast the algorithm converges to the minima.



High learning rate



Low learning rate

To update a_0 and a_1, we take gradients from the cost function. To find these gradients, we take partial derivatives for a_0 and a_1.

$$J = \frac{1}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)^2$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i) \cdot x_i$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

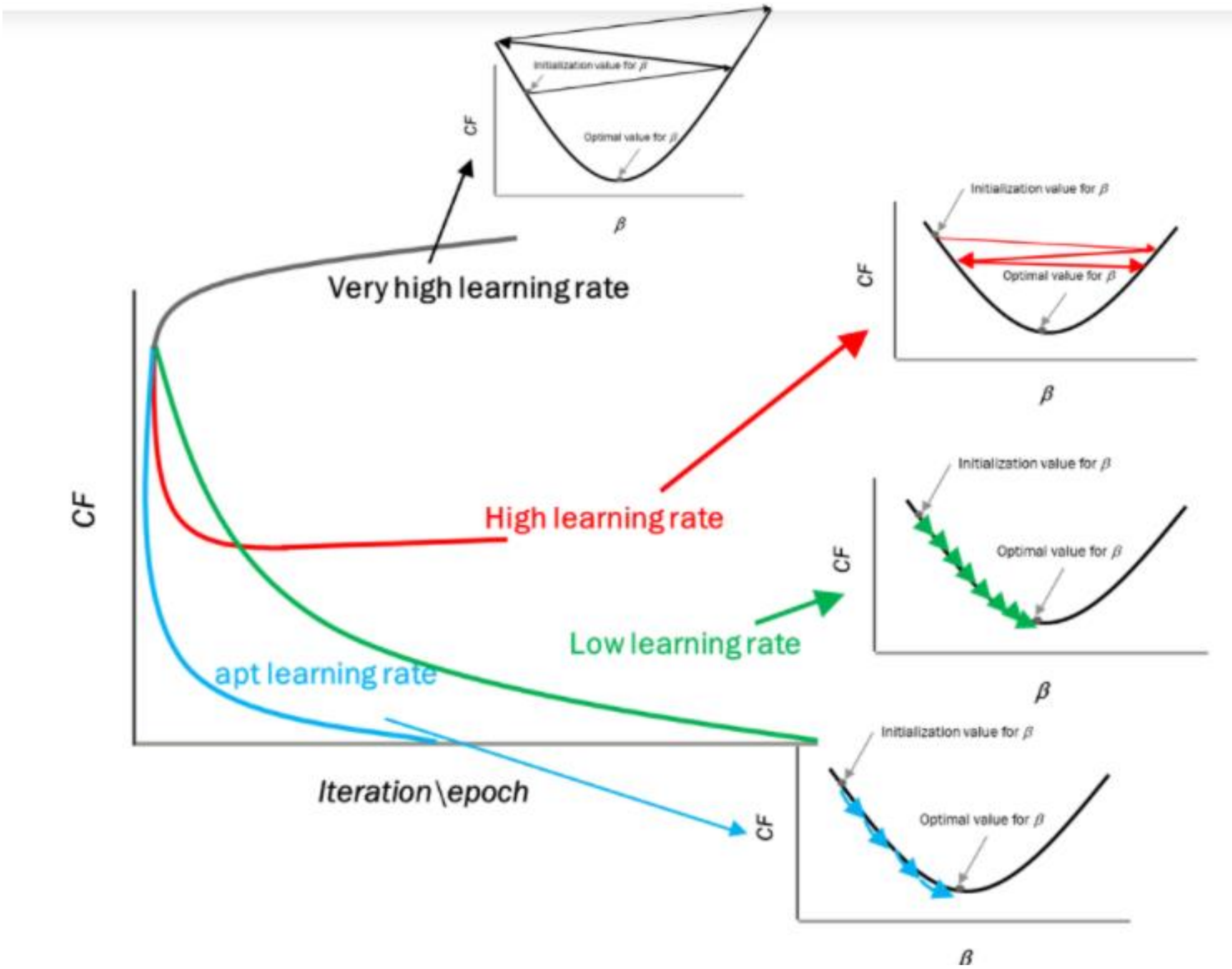
$$a_0 = a_0 - \alpha \cdot \frac{2}{n} \sum_{i=1}^{n} (pred_i - y_i)$$

$$a_1 = a_1 - \alpha \cdot \frac{2}{n} \sum_{i=1}^{n} (pred_i - y_i) \cdot x_i$$

partial derivates are the gradients and they are used to update the

The partial derivates are the gradients, and they are used to update the values of a_0 and a_1. Alpha is the learning rate.

Impact of different values for learning rate



The blue line represents the optimal value of the learning rate, and the cost function value is minimized in a few iterations. The green line represents if the learning rate is lower than the optimal value, then the number of iterations required high to minimize the cost function. If the learning rate selected is very high, the cost function could continue to increase with iterations and saturate at a value higher than the minimum value, that represented by a red and black line.

Decision Tree:

Decision trees are powerful and versatile algorithms used in machine learning for both classification and regression tasks. They are widely employed due to their interpretability, effectiveness, and ability to handle various types of data.

At its core, a decision tree is a flowchart-like structure composed of internal nodes, branches, and leaf nodes. Each internal node represents a feature or attribute, and each branch represents a decision based on that feature. The leaf nodes, located at the ends of the branches, represent the outcomes or predictions.

The process of building a decision tree starts with a labeled dataset, where each example is characterized by a set of input features and a corresponding target variable. The algorithm selects the most informative features by evaluating different criteria, such as information gain, Gini impurity, or entropy. The chosen feature becomes the root node of the decision tree.

The dataset is then split into subsets based on different attribute values. The algorithm identifies the best splitting point that maximizes information gain or minimizes impurity measures. For categorical features, the dataset is divided into subsets based on the distinct attribute values. For numerical features, the algorithm determines the optimal threshold to separate the data.

The splitting process is performed recursively for each subset, generating new nodes and branches in the decision tree. The algorithm continues to evaluate different splitting criteria to determine the best attribute and value at each node. This recursive process persists until a stopping criterion is met. Stopping criteria may include reaching a maximum depth of the tree, a minimum number of samples in a node, or achieving a desired level of purity or accuracy.

After the recursive splitting is completed, the algorithm assigns predictions or outcomes to the leaf nodes. Classification tasks assign the majority class of the samples in a leaf node as the prediction, while regression tasks assign the average value. These predictions represent the results of the decision tree for a given set of input features.

To make predictions for new, unseen data, the input features traverse the decision tree by following the decisions at each node. The traversal continues until a leaf node is reached, and the prediction at that leaf node is used as the output of the decision tree.

One of the key advantages of decision trees is their interpretability. The resulting tree structure can be easily visualized and understood, resembling a flowchart. Decision trees allow analysts and stakeholders to comprehend the decision-making process and gain insights into the factors influencing predictions.

Decision trees can handle both numerical and categorical features, making them applicable to a wide range of datasets. They are also robust to outliers and missing data, as the splitting process is based on relative comparisons rather than absolute values.

However, decision trees are susceptible to overfitting if not properly controlled. They may become too complex and memorize the training data, leading to poor generalization to unseen data. Techniques like pruning, which removes unnecessary nodes or branches, or employing ensemble methods like random forests, can help mitigate overfitting and enhance the performance of decision trees.

In conclusion, decision trees are valuable tools in machine learning, offering a flexible and interpretable approach for classification and regression tasks. Their ability to handle various data types, interpretability, and robustness to outliers make them widely utilized in diverse domains. By constructing a decision tree, analysts and data scientists can gain insights and make informed predictions based on the data's characteristics and patterns.

Codes

```python
#yahoo finance as data source
import yfinance as yf

#See the yahoo finance ticker for the stock symbol
stock_symbol = 'GAIL.NS'

#last 5 years data with interval of 1 day
data = yf.download(tickers=stock_symbol,period='5y',interval='1d')
```

```
[*********************100%***********************]  1 of 1 completed
```

```python
import pandas as pd
type(data)
```

```
pandas.core.frame.DataFrame
```

```python
data.head(10)
```

```
                  Open        High         Low       Close   Adj Close  \
Date
2018-07-16  118.933334  119.833336  116.766663  118.599998   94.274284
2018-07-17  119.333336  122.116669  119.000000  119.566666   95.042686
2018-07-18  120.183334  121.849998  118.433334  119.683334   95.135422
2018-07-19  120.133331  121.349998  118.583336  119.266663   94.804199
2018-07-20  119.599998  120.933334  119.216667  119.933334   95.334145
2018-07-23  120.466667  121.650002  118.783333  119.599998   95.069183
2018-07-24  119.633331  123.099998  118.466667  121.933334   96.923927
2018-07-25  123.000000  123.516663  120.733330  121.550003   96.619232
2018-07-26  121.550003  124.083336  120.966667  123.333336   98.036774
2018-07-27  123.716667  126.599998  122.949997  126.166664  100.288963

              Volume
Date
2018-07-16   8057787
2018-07-17  10419621
2018-07-18  10709484
2018-07-19   5064312
2018-07-20   5851905
2018-07-23   9013209
2018-07-24  13471506
2018-07-25   8770071
2018-07-26  25113513
2018-07-27  10968603
```

```python
len(data)
```

```
1234
```

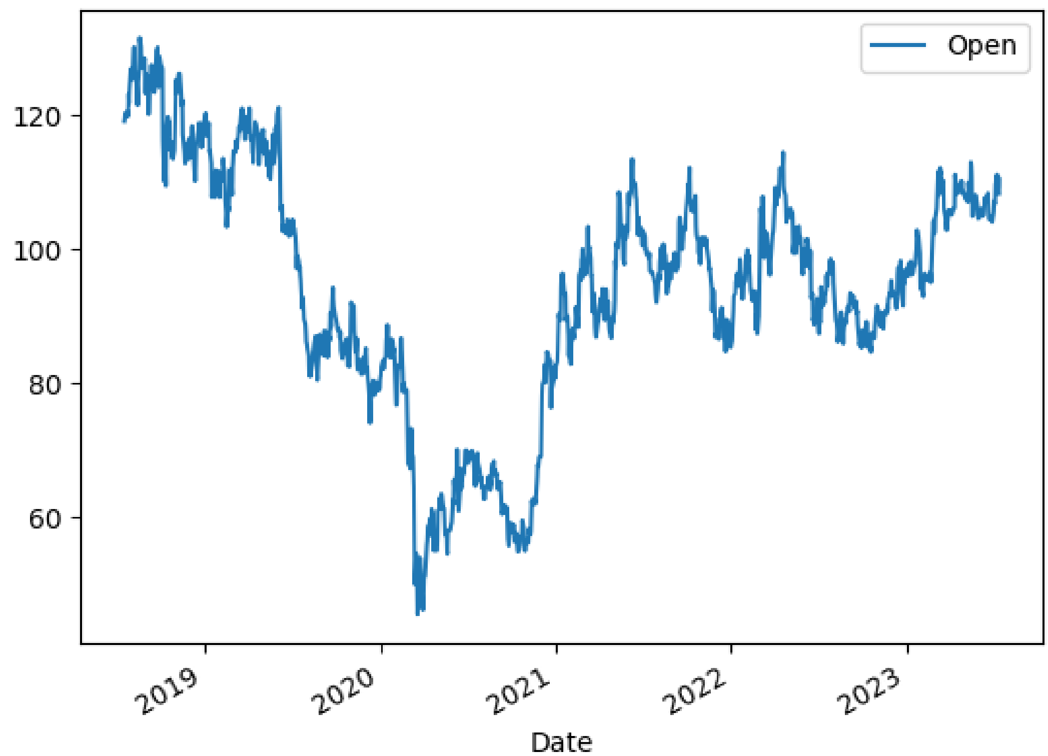
```python
data.tail(10)
```

```
                  Open        High         Low       Close   Adj Close  \
Date
2023-07-03  105.500000  106.599998  104.699997  106.449997  106.449997
2023-07-04  107.000000  107.150002  105.900002  106.550003  106.550003
2023-07-05  106.599998  107.550003  106.150002  107.400002  107.400002
2023-07-06  107.349998  111.400002  106.099998  110.800003  110.800003
2023-07-07  109.599998  112.099998  109.150002  110.699997  110.699997
2023-07-10  111.000000  111.349998  108.300003  108.949997  108.949997
2023-07-11  108.949997  109.599998  107.650002  109.300003  109.300003
2023-07-12  109.000000  111.150002  108.949997  110.599998  110.599998
2023-07-13  110.599998  111.199997  107.849998  108.099998  108.099998
2023-07-14  108.199997  110.599998  107.500000  110.000000  110.000000

              Volume
Date
2023-07-03  22772148
2023-07-04   9292916
2023-07-05  10903523
2023-07-06  29145409
2023-07-07  15022279
2023-07-10   8710335
2023-07-11  10380385
2023-07-12   9851130
2023-07-13   8440163
2023-07-14   7518676
```

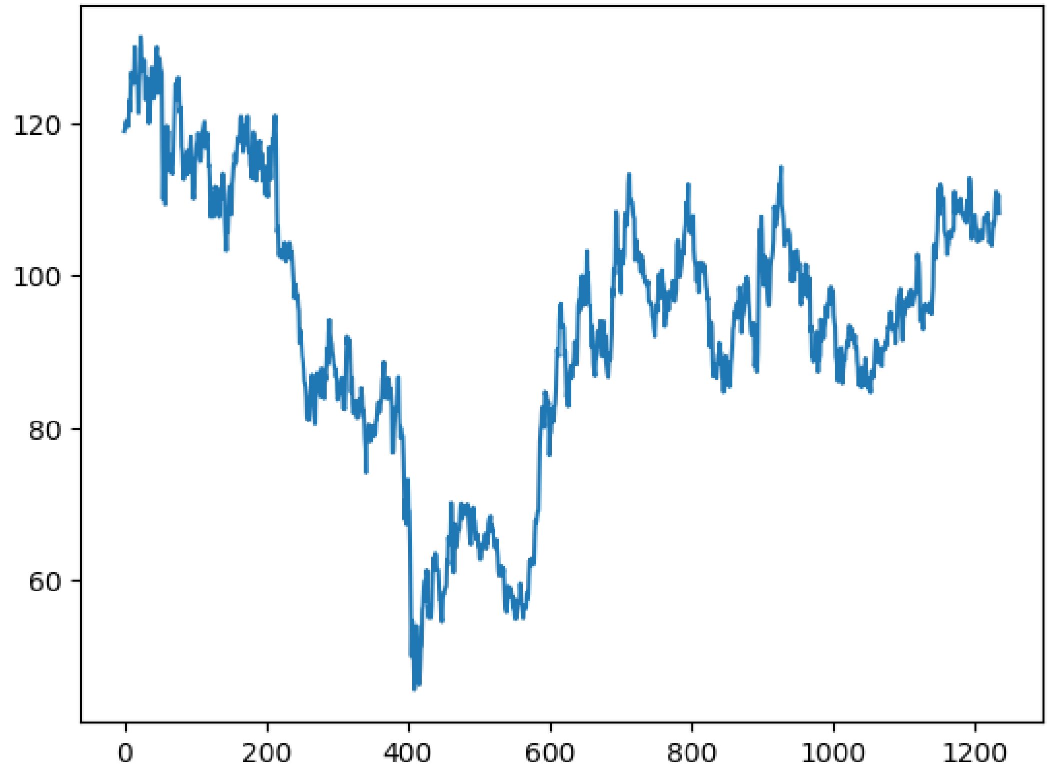
```python
opn = data[['Open']]
```

```python
opn.plot()
```

```
<Axes: xlabel='Date'>
```




```
import numpy as np
import matplotlib.pyplot as plt

ds = opn.values

ds

array([[118.93333435],
       [119.33333588],
       [120.18333435],
       ...,
       [109.        ],
       [110.59999847],
       [108.19999695]])

plt.plot(ds)

[<matplotlib.lines.Line2D at 0x79b097c80dc0>]
```




```
from sklearn.preprocessing import MinMaxScaler

#Using MinMaxScaler for normalizing data between 0 & 1
normalizer = MinMaxScaler(feature_range=(0,1))
ds_scaled = normalizer.fit_transform(np.array(ds).reshape(-1,1))

len(ds_scaled), len(ds)

(1234, 1234)

#Defining test and train data sizes
train_size = int(len(ds_scaled)*0.70)
test_size = len(ds_scaled) - train_size

train_size,test_size

(863, 371)

#Splitting data between train and test
ds_train, ds_test = ds_scaled[0:train_size,:],
ds_scaled[train_size:len(ds_scaled),:1]

len(ds_train),len(ds_test)

(863, 371)

#creating dataset in time series for LSTM model
#X[100,120,140,160,180] : Y[200]
```

```python
def create_ds(dataset,step):
    Xtrain, Ytrain = [], []
    for i in range(len(dataset)-step-1):
        a = dataset[i:(i+step), 0]
        Xtrain.append(a)
        Ytrain.append(dataset[i + step, 0])
    return np.array(Xtrain), np.array(Ytrain)

#Taking 100 days price as one record for training
time_stamp = 100
X_train, y_train = create_ds(ds_train,time_stamp)
X_test, y_test = create_ds(ds_test,time_stamp)

X_train.shape,y_train.shape

((762, 100), (762,))

X_test.shape, y_test.shape

((270, 100), (270,))

#Reshaping data to fit into LSTM model
X_train = X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

from keras.models import Sequential
from keras.layers import Dense, LSTM

#Creating LSTM model using keras
model = Sequential()
model.add(LSTM(units=50,return_sequences=True,input_shape=(X_train.shape[1],1
)))
model.add(LSTM(units=50,return_sequences=True))
model.add(LSTM(units=50))
model.add(Dense(units=1,activation='linear'))
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 lstm_1 (LSTM)               (None, 100, 50)           20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50,851
```

```
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

```
#Training model with adam optimizer and mean squared error loss function
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100,batch_si
ze=64)
```

```
Epoch 1/100
12/12 [==============================] - 6s 195ms/step - loss: 0.0903 -
val_loss: 0.0077
Epoch 2/100
12/12 [==============================] - 1s 123ms/step - loss: 0.0151 -
val_loss: 0.0045
Epoch 3/100
12/12 [==============================] - 2s 177ms/step - loss: 0.0087 -
val_loss: 0.0067
Epoch 4/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0058 -
val_loss: 0.0020
Epoch 5/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0044 -
val_loss: 0.0020
Epoch 6/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0040 -
val_loss: 0.0023
Epoch 7/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0039 -
val_loss: 0.0020
Epoch 8/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0038 -
val_loss: 0.0019
Epoch 9/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0037 -
val_loss: 0.0021
Epoch 10/100
12/12 [==============================] - 2s 196ms/step - loss: 0.0036 -
val_loss: 0.0019
Epoch 11/100
12/12 [==============================] - 2s 162ms/step - loss: 0.0035 -
val_loss: 0.0018
Epoch 12/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0033 -
val_loss: 0.0019
Epoch 13/100
12/12 [==============================] - 1s 118ms/step - loss: 0.0032 -
val_loss: 0.0019
Epoch 14/100
12/12 [==============================] - 2s 164ms/step - loss: 0.0031 -
```

```
val_loss: 0.0019
Epoch 15/100
12/12 [==============================] - 2s 141ms/step - loss: 0.0030 -
val_loss: 0.0022
Epoch 16/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0030 -
val_loss: 0.0019
Epoch 17/100
12/12 [==============================] - 2s 134ms/step - loss: 0.0029 -
val_loss: 0.0017
Epoch 18/100
12/12 [==============================] - 2s 159ms/step - loss: 0.0029 -
val_loss: 0.0020
Epoch 19/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0028 -
val_loss: 0.0017
Epoch 20/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0028 -
val_loss: 0.0031
Epoch 21/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0028 -
val_loss: 0.0017
Epoch 22/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0027 -
val_loss: 0.0022
Epoch 23/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0027 -
val_loss: 0.0015
Epoch 24/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0025 -
val_loss: 0.0015
Epoch 25/100
12/12 [==============================] - 2s 145ms/step - loss: 0.0027 -
val_loss: 0.0019
Epoch 26/100
12/12 [==============================] - 2s 145ms/step - loss: 0.0026 -
val_loss: 0.0022
Epoch 27/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0024 -
val_loss: 0.0014
Epoch 28/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0024 -
val_loss: 0.0013
Epoch 29/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0023 -
val_loss: 0.0013
Epoch 30/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0024 -
val_loss: 0.0013
Epoch 31/100
```

```
12/12 [==============================] - 1s 120ms/step - loss: 0.0025 -
val_loss: 0.0013
Epoch 32/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0023 -
val_loss: 0.0013
Epoch 33/100
12/12 [==============================] - 2s 158ms/step - loss: 0.0021 -
val_loss: 0.0012
Epoch 34/100
12/12 [==============================] - 2s 131ms/step - loss: 0.0022 -
val_loss: 0.0012
Epoch 35/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0020 -
val_loss: 0.0014
Epoch 36/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0020 -
val_loss: 0.0011
Epoch 37/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0019 -
val_loss: 0.0011
Epoch 38/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0019 -
val_loss: 0.0013
Epoch 39/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0019 -
val_loss: 0.0013
Epoch 40/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0019 -
val_loss: 0.0019
Epoch 41/100
12/12 [==============================] - 2s 175ms/step - loss: 0.0020 -
val_loss: 0.0011
Epoch 42/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0019 -
val_loss: 0.0010
Epoch 43/100
12/12 [==============================] - 1s 118ms/step - loss: 0.0018 -
val_loss: 0.0018
Epoch 44/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0018 -
val_loss: 8.8369e-04
Epoch 45/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0017 -
val_loss: 8.7292e-04
Epoch 46/100
12/12 [==============================] - 1s 123ms/step - loss: 0.0016 -
val_loss: 8.6852e-04
Epoch 47/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0016 -
val_loss: 9.4270e-04
```

```
Epoch 48/100
12/12 [==============================] - 1s 124ms/step - loss: 0.0016 -
val_loss: 0.0011
Epoch 49/100
12/12 [==============================] - 2s 168ms/step - loss: 0.0016 -
val_loss: 0.0014
Epoch 50/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0019 -
val_loss: 0.0015
Epoch 51/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0017 -
val_loss: 7.7475e-04
Epoch 52/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0015 -
val_loss: 8.0506e-04
Epoch 53/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0015 -
val_loss: 8.4291e-04
Epoch 54/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0014 -
val_loss: 0.0012
Epoch 55/100
12/12 [==============================] - 1s 124ms/step - loss: 0.0015 -
val_loss: 8.1491e-04
Epoch 56/100
12/12 [==============================] - 2s 143ms/step - loss: 0.0014 -
val_loss: 9.2286e-04
Epoch 57/100
12/12 [==============================] - 2s 147ms/step - loss: 0.0015 -
val_loss: 0.0012
Epoch 58/100
12/12 [==============================] - 1s 123ms/step - loss: 0.0015 -
val_loss: 7.0306e-04
Epoch 59/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0013 -
val_loss: 8.0683e-04
Epoch 60/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0013 -
val_loss: 7.0186e-04
Epoch 61/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0013 -
val_loss: 0.0014
Epoch 62/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0017 -
val_loss: 0.0010
Epoch 63/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0017 -
val_loss: 0.0013
Epoch 64/100
12/12 [==============================] - 2s 162ms/step - loss: 0.0015 -
```

```
val_loss: 7.4070e-04
Epoch 65/100
12/12 [==============================] - 2s 132ms/step - loss: 0.0013 -
val_loss: 6.5428e-04
Epoch 66/100
12/12 [==============================] - 1s 123ms/step - loss: 0.0012 -
val_loss: 7.8155e-04
Epoch 67/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0012 -
val_loss: 6.6853e-04
Epoch 68/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0013 -
val_loss: 6.3991e-04
Epoch 69/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0013 -
val_loss: 9.0017e-04
Epoch 70/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0012 -
val_loss: 6.1420e-04
Epoch 71/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0012 -
val_loss: 7.6780e-04
Epoch 72/100
12/12 [==============================] - 2s 180ms/step - loss: 0.0011 -
val_loss: 6.1783e-04
Epoch 73/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0011 -
val_loss: 5.9184e-04
Epoch 74/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0011 -
val_loss: 6.0342e-04
Epoch 75/100
12/12 [==============================] - 1s 124ms/step - loss: 0.0011 -
val_loss: 5.8076e-04
Epoch 76/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0012 -
val_loss: 7.1327e-04
Epoch 77/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0011 -
val_loss: 6.2529e-04
Epoch 78/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0011 -
val_loss: 5.5967e-04
Epoch 79/100
12/12 [==============================] - 1s 127ms/step - loss: 0.0011 -
val_loss: 5.5552e-04
Epoch 80/100
12/12 [==============================] - 2s 165ms/step - loss: 0.0010 -
val_loss: 6.5665e-04
Epoch 81/100
```

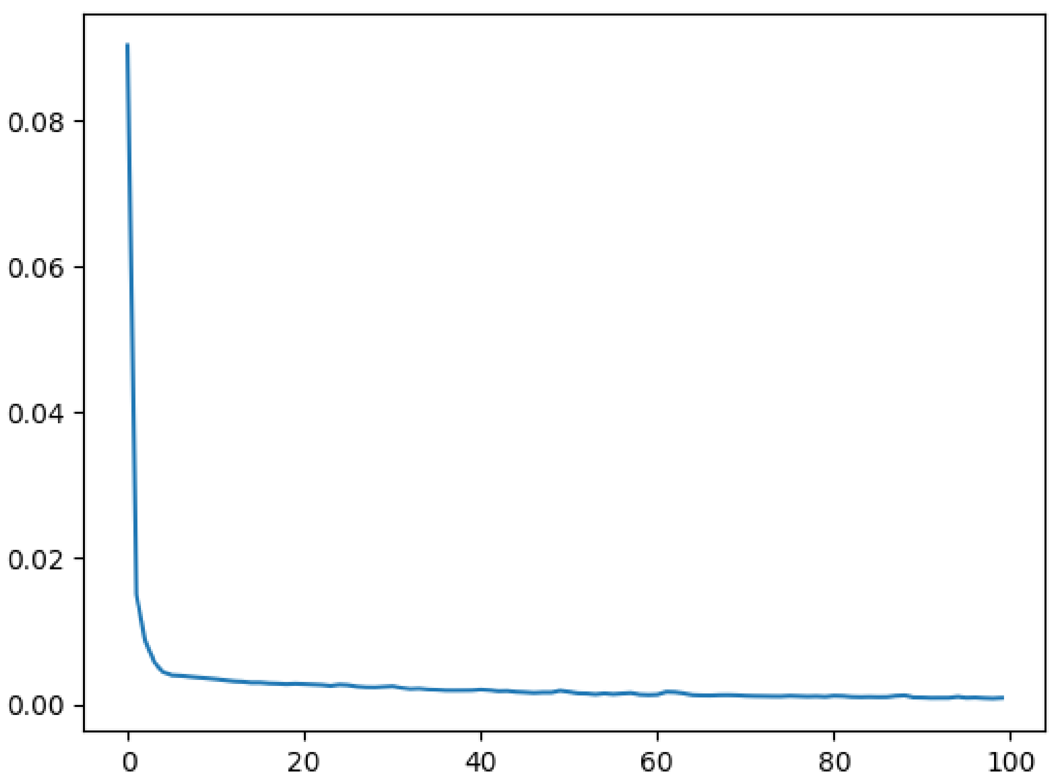
```
12/12 [==============================] - 1s 122ms/step - loss: 0.0012 -
val_loss: 7.8654e-04
Epoch 82/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0011 -
val_loss: 5.4887e-04
Epoch 83/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0010 -
val_loss: 5.4196e-04
Epoch 84/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0010 -
val_loss: 6.5397e-04
Epoch 85/100
12/12 [==============================] - 1s 123ms/step - loss: 0.0010 -
val_loss: 7.3919e-04
Epoch 86/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0010 -
val_loss: 5.1618e-04
Epoch 87/100
12/12 [==============================] - 2s 147ms/step - loss: 0.0010 -
val_loss: 5.2707e-04
Epoch 88/100
12/12 [==============================] - 2s 147ms/step - loss: 0.0012 -
val_loss: 0.0011
Epoch 89/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0012 -
val_loss: 5.0115e-04
Epoch 90/100
12/12 [==============================] - 1s 119ms/step - loss: 9.6256e-04 -
val_loss: 6.3312e-04
Epoch 91/100
12/12 [==============================] - 1s 120ms/step - loss: 9.3806e-04 -
val_loss: 4.9813e-04
Epoch 92/100
12/12 [==============================] - 1s 119ms/step - loss: 8.8895e-04 -
val_loss: 4.8711e-04
Epoch 93/100
12/12 [==============================] - 1s 121ms/step - loss: 8.9661e-04 -
val_loss: 4.8865e-04
Epoch 94/100
12/12 [==============================] - 1s 122ms/step - loss: 8.9853e-04 -
val_loss: 6.6910e-04
Epoch 95/100
12/12 [==============================] - 2s 161ms/step - loss: 0.0010 -
val_loss: 4.9413e-04
Epoch 96/100
12/12 [==============================] - 2s 130ms/step - loss: 9.0525e-04 -
val_loss: 6.8587e-04
Epoch 97/100
12/12 [==============================] - 1s 124ms/step - loss: 9.4197e-04 -
val_loss: 4.6827e-04
```

```
Epoch 98/100
12/12 [==============================] - 1s 122ms/step - loss: 8.6799e-04 -
val_loss: 4.6517e-04
Epoch 99/100
12/12 [==============================] - 1s 120ms/step - loss: 8.2689e-04 -
val_loss: 4.6597e-04
Epoch 100/100
12/12 [==============================] - 1s 122ms/step - loss: 8.9514e-04 -
val_loss: 7.3215e-04
```

```
<keras.callbacks.History at 0x79b034766aa0>
```

```
#PLotting loss, it shows that loss has decreased significantly and model
trained well
loss = model.history.history['loss']
plt.plot(loss)
```

```
[<matplotlib.lines.Line2D at 0x79b023c22710>]
```



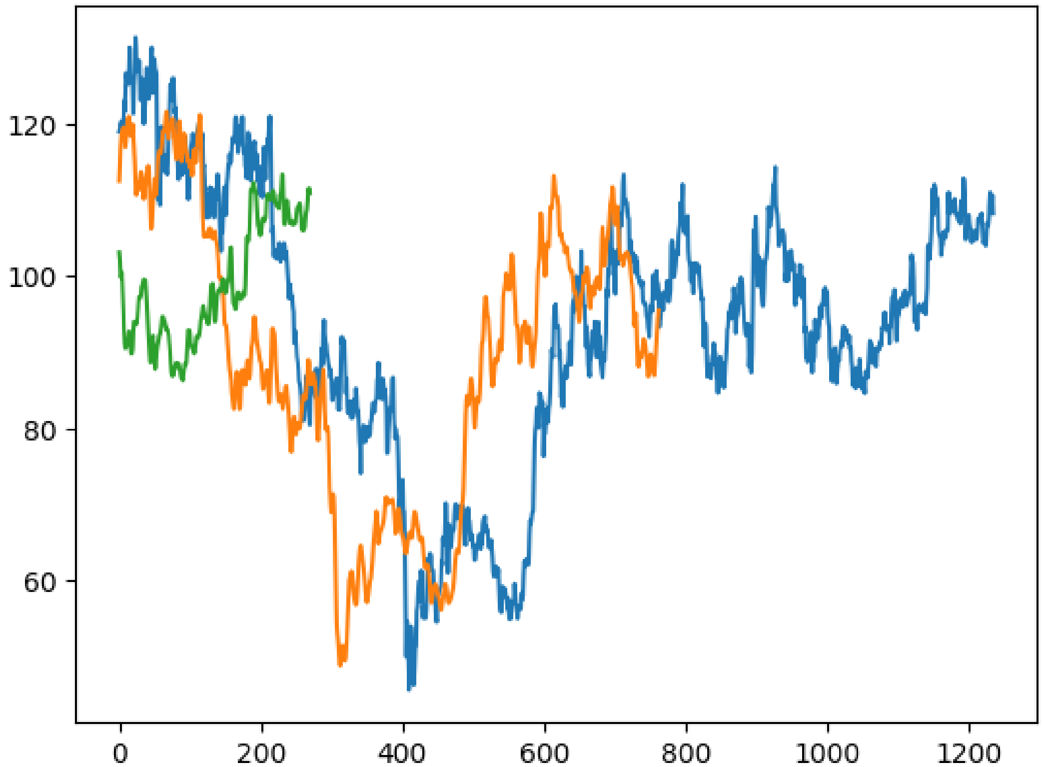

```
#Predicitng on train and test data
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

```
24/24 [==============================] - 1s 26ms/step
9/9 [==============================] - 0s 30ms/step
```

```
#Inverse transform to get actual value
train_predict = normalizer.inverse_transform(train_predict)
test_predict = normalizer.inverse_transform(test_predict)
```

```
#Comparing using visuals
plt.plot(normalizer.inverse_transform(ds_scaled))
plt.plot(train_predict)
plt.plot(test_predict)
```

```
[<matplotlib.lines.Line2D at 0x79b03501b790>]
```



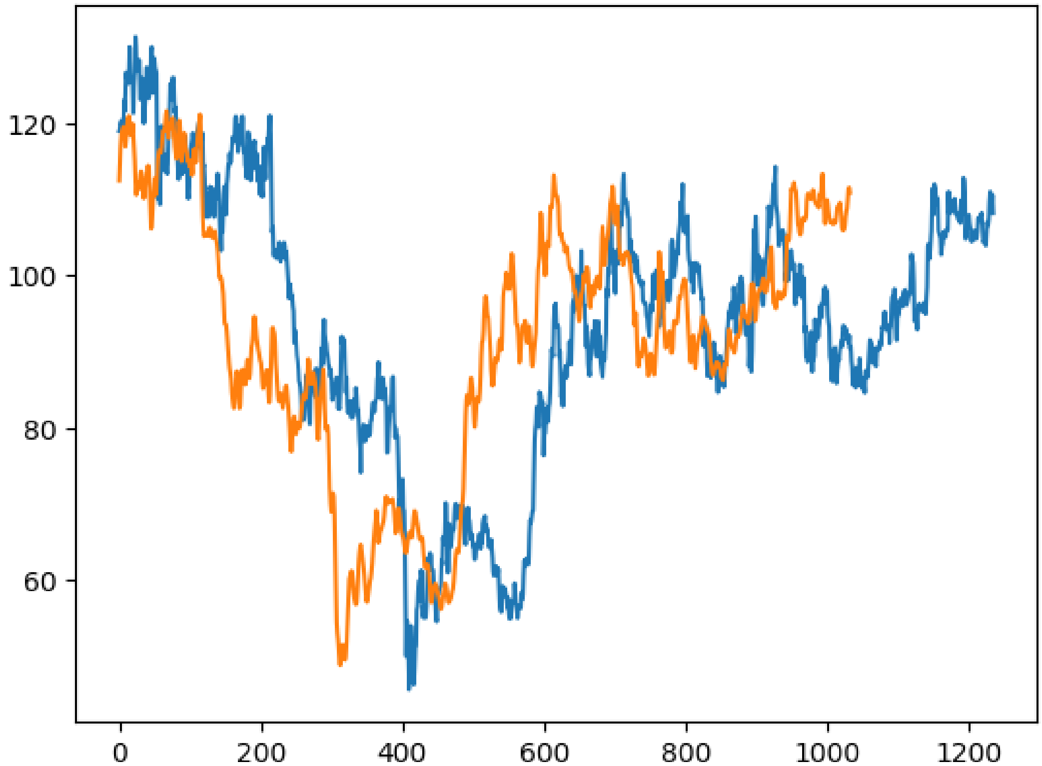

```
type(train_predict)
```

```
numpy.ndarray
```

```
test = np.vstack((train_predict,test_predict))
```

```
#Combining the predited data to create uniform data visualization
plt.plot(normalizer.inverse_transform(ds_scaled))
plt.plot(test)
```

```
[<matplotlib.lines.Line2D at 0x79b034f26650>]
```




```
len(ds_test)
```

```
371
```

```
#Getting the last 100 days records
fut_inp = ds_test[270:]

fut_inp = fut_inp.reshape(1,-1)

tmp_inp = list(fut_inp)

fut_inp.shape
```

```
(1, 101)
```

```
#Creating list of the last 100 data
tmp_inp = tmp_inp[0].tolist()

#Predicting next 30 days price suing the current data
#It will predict in sliding window manner (algorithm) with stride 1
lst_output=[]
n_steps=100
i=0
while(i<30):

    if(len(tmp_inp)>100):
        fut_inp = np.array(tmp_inp[1:])
```

```
        fut_inp=fut_inp.reshape(1,-1)
        fut_inp = fut_inp.reshape((1, n_steps, 1))
        yhat = model.predict(fut_inp, verbose=0)
        tmp_inp.extend(yhat[0].tolist())
        tmp_inp = tmp_inp[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        fut_inp = fut_inp.reshape((1, n_steps,1))
        yhat = model.predict(fut_inp, verbose=0)
        tmp_inp.extend(yhat[0].tolist())
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
```

```
[[0.7580356001853943], [0.762689471244812], [0.7723258137702942],
[0.7826582789421082], [0.7920875549316406], [0.8006947040557861],
[0.8088741302490234], [0.816780149936676], [0.8243721127510071],
[0.83158278465271], [0.8383971452713013], [0.8448458313941956],
[0.8509761691093445], [0.8568329215049744], [0.862455427646637],
[0.8678773045539856], [0.8731271028518677], [0.878229558467865],
[0.8832032680511475], [0.8880634307861328], [0.8928189873695374],
[0.897477924823761], [0.902043342590332], [0.9065182209014893],
[0.910903811454773], [0.9152010679244995], [0.919410228729248],
[0.9235321283340454], [0.9275680780410767], [0.9315181970596313]]
```

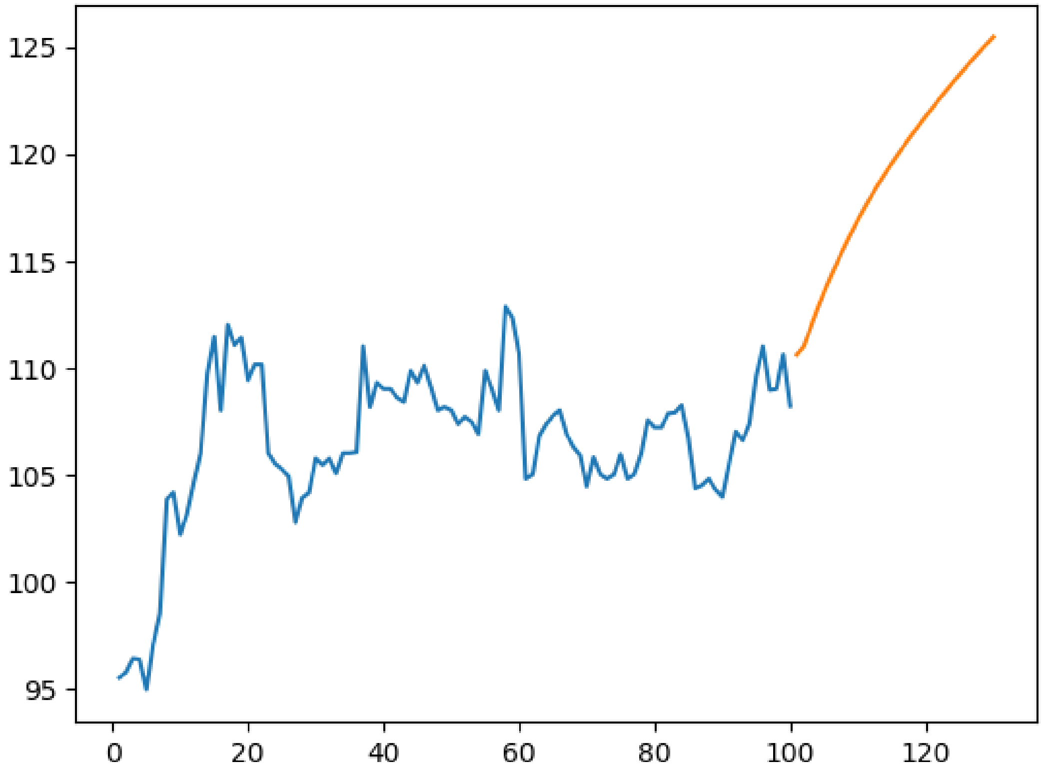
```
len(ds_scaled)
```

```
1234
```

```
#Creating a dummy plane to plot graph one after another
plot_new=np.arange(1,101)
plot_pred=np.arange(101,131)
```

```
plt.plot(plot_new, normalizer.inverse_transform(ds_scaled[1134:]))
plt.plot(plot_pred, normalizer.inverse_transform(lst_output))
```

```
[<matplotlib.lines.Line2D at 0x79b034dbe290>]
```



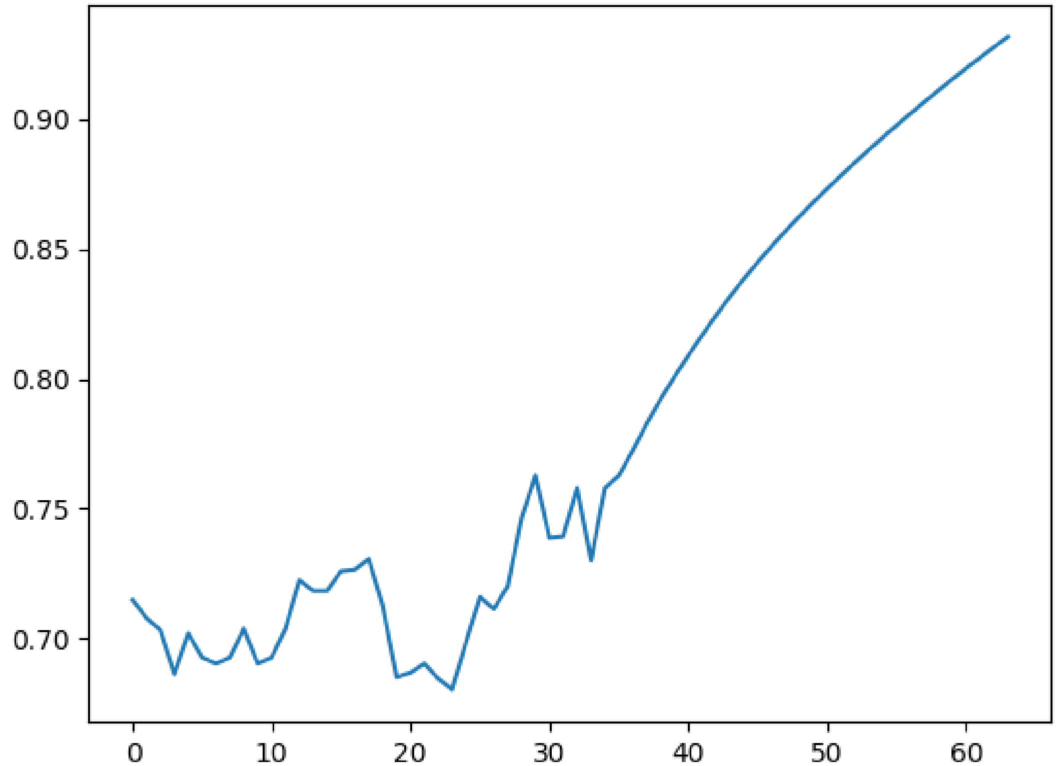

```
ds_new = ds_scaled.tolist()
```

```
len(ds_new)
```

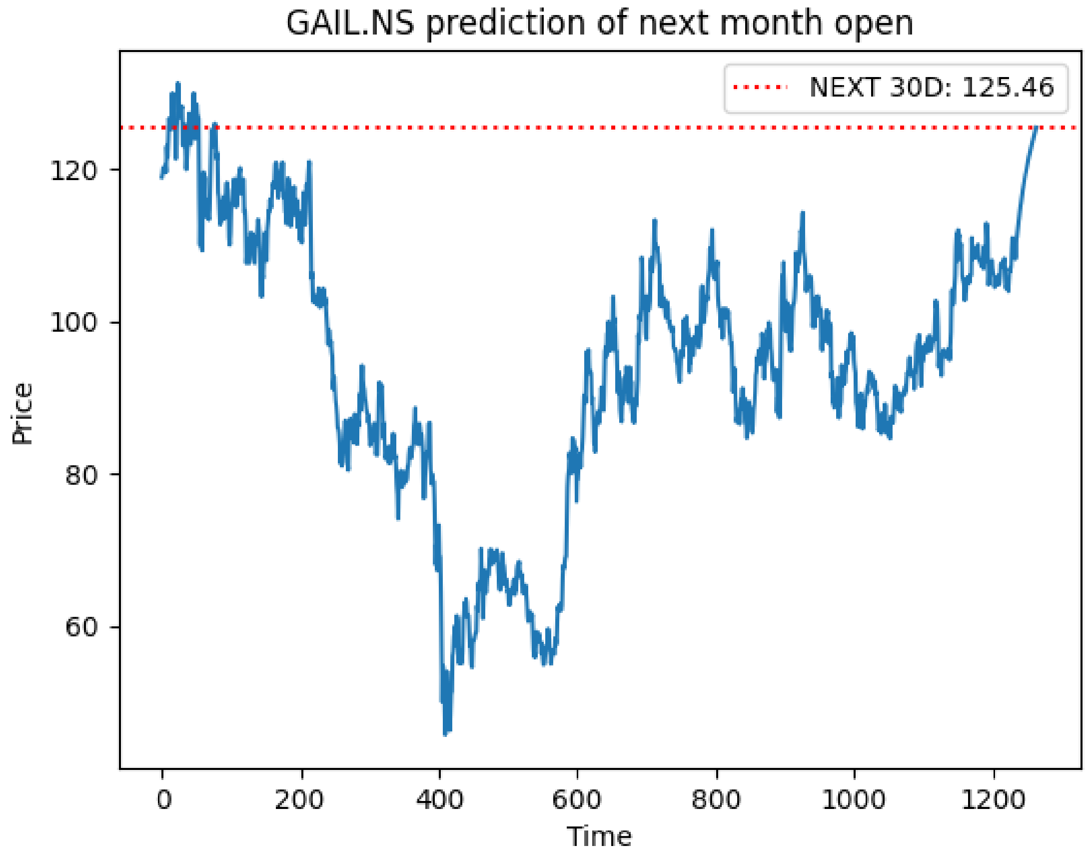
1234

```
#Entends helps us to fill the missing value with approx value
ds_new.extend(lst_output)
plt.plot(ds_new[1200:])
```

[<matplotlib.lines.Line2D at 0x79b034c8c100>]




```
#Creating final data for plotting
final_graph = normalizer.inverse_transform(ds_new).tolist()

#Plotting final results with predicted value after 30 Days
plt.plot(final_graph,)
plt.ylabel("Price")
plt.xlabel("Time")
plt.title("{0} prediction of next month open".format(stock_symbol))
plt.axhline(y=final_graph[len(final_graph)-1], color = 'red', linestyle =
':', label = 'NEXT 30D:
{0}'.format(round(float(*final_graph[len(final_graph)-1]),2)))
plt.legend()

<matplotlib.legend.Legend at 0x79b034cb3f10>
```


GAIL.NS prediction of next month open
NEXT 30D: 125.46
Price
Time
120
100
80
60
0
200
400
600
800
1000
1200

For Comparing Models:

Importing required Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
import io
```

Loading the dataset

```
from google.colab import files

uploaded = files.upload()
```

```
<IPython.core.display.HTML object>

Saving RELIANCE.csv to RELIANCE.csv
```

```
data = pd.read_csv(io.BytesIO(uploaded['RELIANCE.csv']))
```

Show head of the dataset

```
print("Head of the dataset:")
print(data.head())
```

```
Head of the dataset:
         Date         Open         High          Low        Close  \
0  2022-07-12  2404.000000  2439.699951  2404.000000  2420.449951
1  2022-07-13  2427.300049  2434.000000  2373.000000  2377.550049
2  2022-07-14  2388.000000  2433.949951  2376.949951  2397.149902
3  2022-07-15  2415.000000  2415.000000  2383.100098  2401.800049
4  2022-07-18  2421.000000  2425.000000  2392.300049  2422.250000

     Adj Close   Volume
0  2413.184570  4974502
1  2370.413330  6564435
2  2389.954346  7831798
3  2394.590576  4431880
4  2414.979248  6996757
```

Show tail of the dataset

```python
print("Tail of the dataset:")
print(data.tail())
```

```
Tail of the dataset:
           Date         Open         High          Low        Close  \
242  2023-07-05  2609.000000  2609.000000  2575.800049  2584.500000
243  2023-07-06  2576.050049  2644.449951  2576.050049  2638.750000
244  2023-07-07  2635.000000  2664.949951  2628.000000  2633.600098
245  2023-07-10  2688.899902  2756.000000  2675.000000  2735.050049
246  2023-07-11  2752.899902  2770.000000  2737.600098  2764.699951

       Adj Close    Volume
242  2584.500000   4729479
243  2638.750000   8822948
244  2633.600098   6172684
245  2735.050049  15340262
246  2764.699951   9250766
```

Check dataset information

```python
print("Dataset Information:")
print(data.info())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 247 entries, 0 to 246
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       247 non-null    object
 1   Open       247 non-null    float64
 2   High       247 non-null    float64
 3   Low        247 non-null    float64
 4   Close      247 non-null    float64
 5   Adj Close  247 non-null    float64
 6   Volume     247 non-null    int64
dtypes: float64(5), int64(1), object(1)
memory usage: 13.6+ KB
None
```

Check for null values

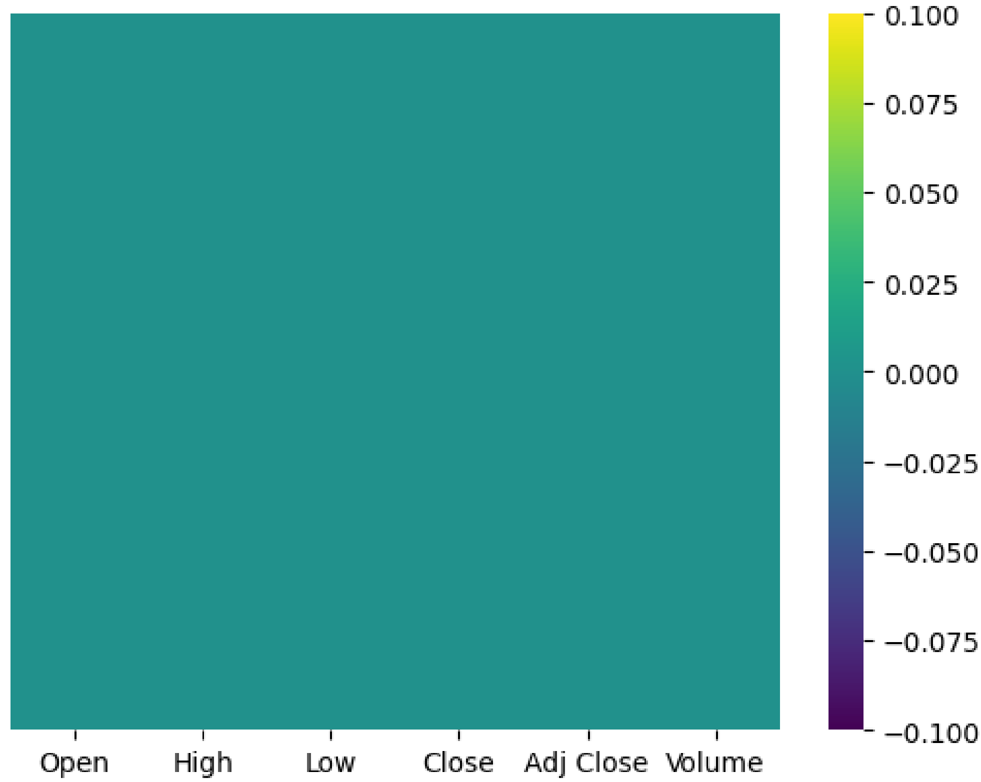
```python
print("Null values:")
print(data.isnull().sum())
```

```
Null values:
Date         0
Open         0
High         0
Low          0
Close        0
```

```
Adj Close    0
Volume       0
dtype: int64
```

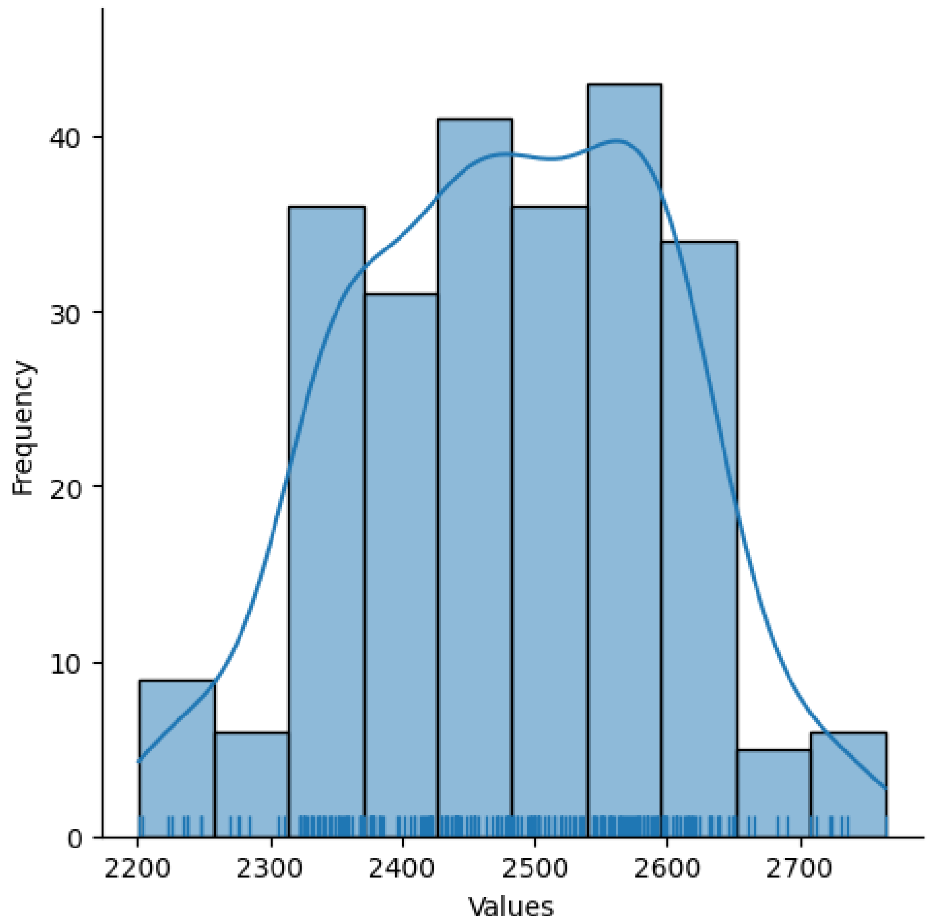
```python
import seaborn as sns
sns.heatmap(data.isnull(),yticklabels=False,cbar=True,cmap="viridis")
```

```
<Axes: >
```



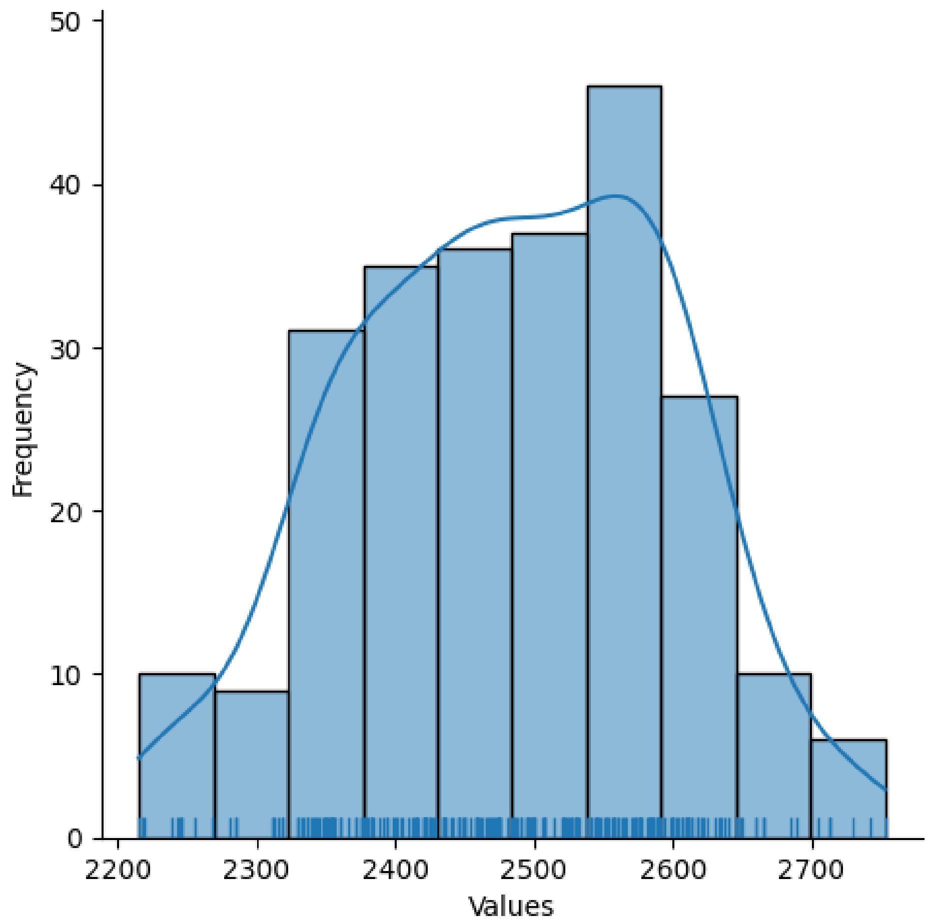

```python
plt.figure(figsize=(10, 6))
sns.displot(data['Close'], kde=True, rug=True)
#plt.title('Histogram with Rug Plot - Outliers')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



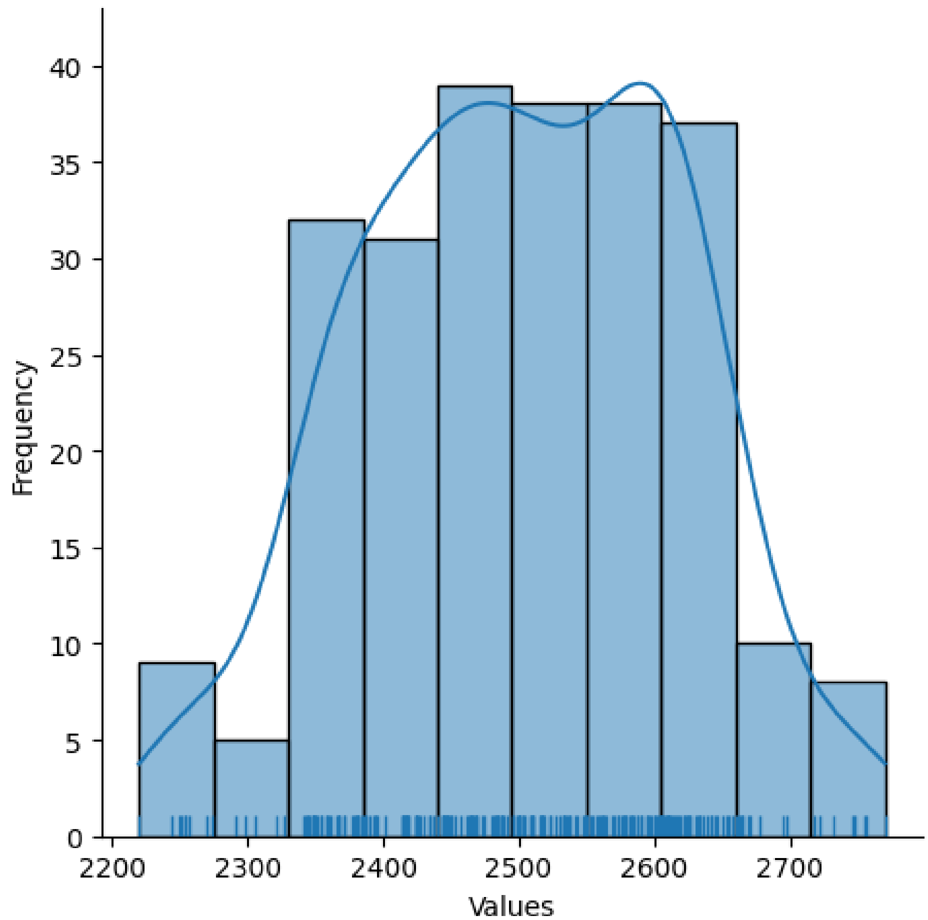

```python
plt.figure(figsize=(10, 6))
sns.displot(data['Open'], kde=True, rug=True)
#plt.title('Histogram with Rug Plot - Outliers')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



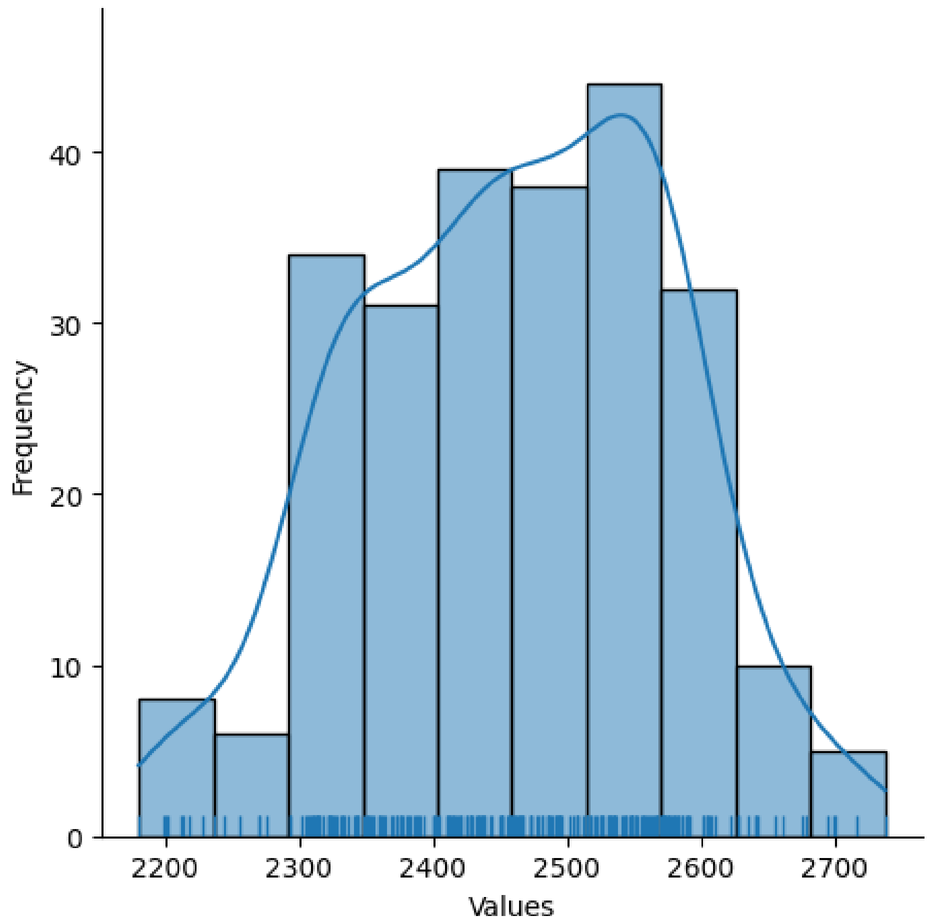

```
plt.figure(figsize=(10, 6))
sns.displot(data['High'], kde=True, rug=True)
#plt.title('Histogram with Rug Plot - Outliers')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```




```
plt.figure(figsize=(10, 6))
sns.displot(data['Low'], kde=True, rug=True)
#plt.title('Histogram with Rug Plot - Outliers')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```




```
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

# Detect outliers using the IQR method
outliers = ((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 *
IQR))).any(axis=1)

# Print rows with outliers
print("Rows with outliers:")
print(data[outliers])
```

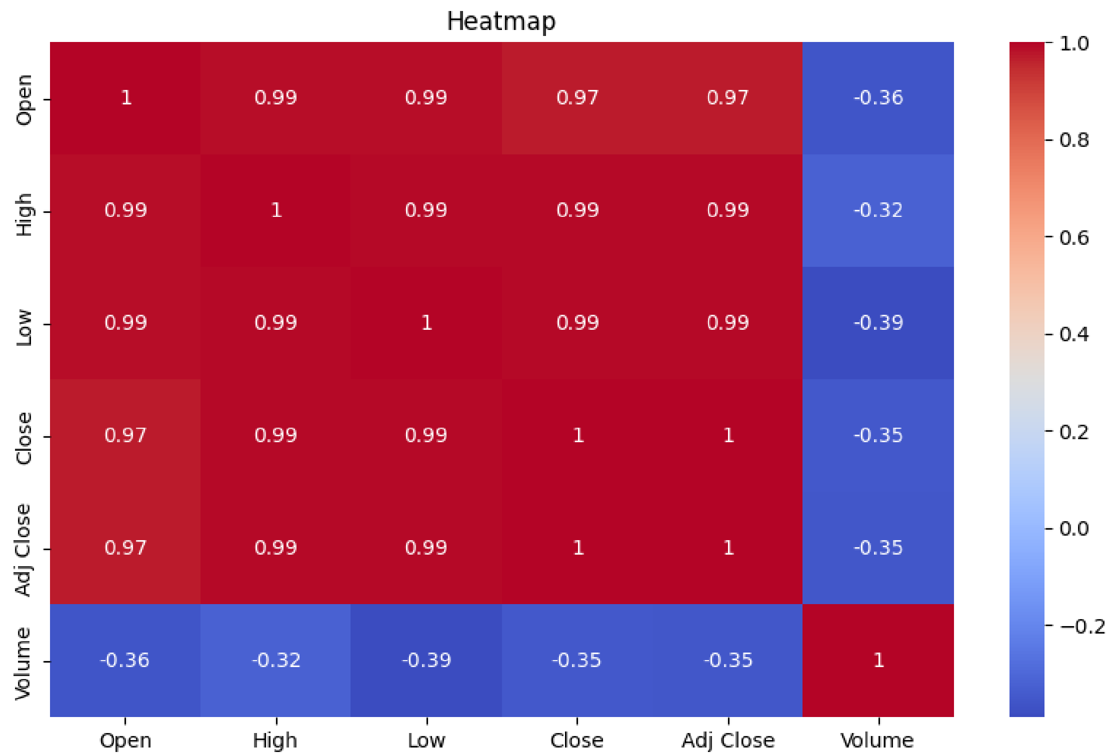
```
Rows with outliers:
            Open         High          Low        Close    Adj Close
Volume
6    2540.000000  2542.500000  2486.250000  2503.000000  2495.486816
11041036
93   2608.899902  2721.050049  2502.000000  2707.550049  2707.550049
14549929
95   2712.500000  2745.449951  2698.199951  2731.350098  2731.350098
12075137
136  2384.399902  2387.350098  2311.649902  2337.350098  2337.350098
```

```
11920991
141  2349.000000  2349.000000  2293.000000  2329.000000  2329.000000
11398850
149  2376.000000  2437.199951  2373.000000  2431.949951  2431.949951
15461902
170  2244.750000  2251.949951  2212.699951  2223.100098  2223.100098
15697554
179  2255.000000  2343.449951  2254.699951  2331.050049  2331.050049
13001005
218  2500.000000  2509.850098  2461.000000  2469.899902  2469.899902
12510304
230  2560.199951  2582.399902  2560.199951  2577.399902  2577.399902
11155180
245  2688.899902  2756.000000  2675.000000  2735.050049  2735.050049
15340262
```

```python
dataT = data[~outliers]

# Print the modified dataset without outliers
print("Modified dataset without outliers:")
print(dataT)
```

```
Modified dataset without outliers:
            Open         High          Low        Close    Adj Close   Volume
0    2404.000000  2439.699951  2404.000000  2420.449951  2413.184570  4974502
1    2427.300049  2434.000000  2373.000000  2377.550049  2370.413330  6564435
2    2388.000000  2433.949951  2376.949951  2397.149902  2389.954346  7831798
3    2415.000000  2415.000000  2383.100098  2401.800049  2394.590576  4431880
4    2421.000000  2425.000000  2392.300049  2422.250000  2414.979248  6996757
..           ...          ...          ...          ...          ...      ...
241  2625.000000  2625.000000  2573.250000  2588.750000  2588.750000  3720447
242  2609.000000  2609.000000  2575.800049  2584.500000  2584.500000  4729479
243  2576.050049  2644.449951  2576.050049  2638.750000  2638.750000  8822948
244  2635.000000  2664.949951  2628.000000  2633.600098  2633.600098  6172684
246  2752.899902  2770.000000  2737.600098  2764.699951  2764.699951  9250766

[236 rows x 6 columns]
```

```python
plt.figure(figsize=(10, 6))
sns.heatmap(dataT.corr(), annot=True, cmap='coolwarm')
plt.title('Heatmap')
plt.show()
```



Normalize the data

```
scaler = MinMaxScaler()
data_normalized = pd.DataFrame(scaler.fit_transform(dataT),
columns=data.columns)
```

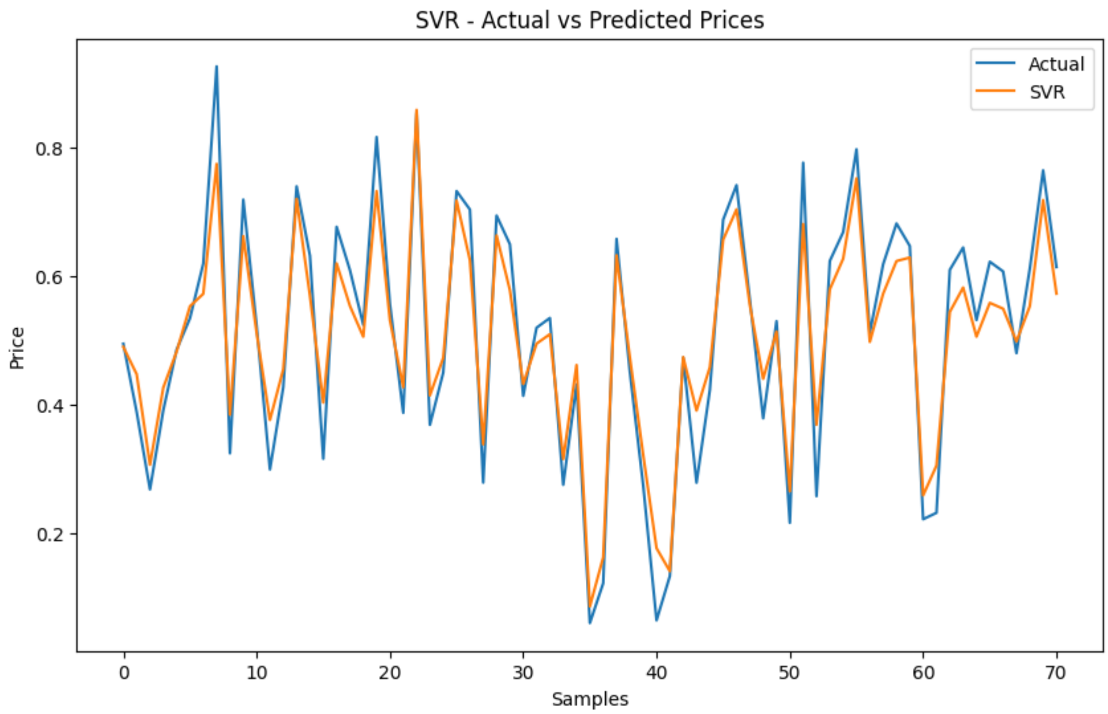
Split the data into training and testing sets

```
X = data_normalized.drop('Close', axis=1)
y = data_normalized['Close']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

SVR model

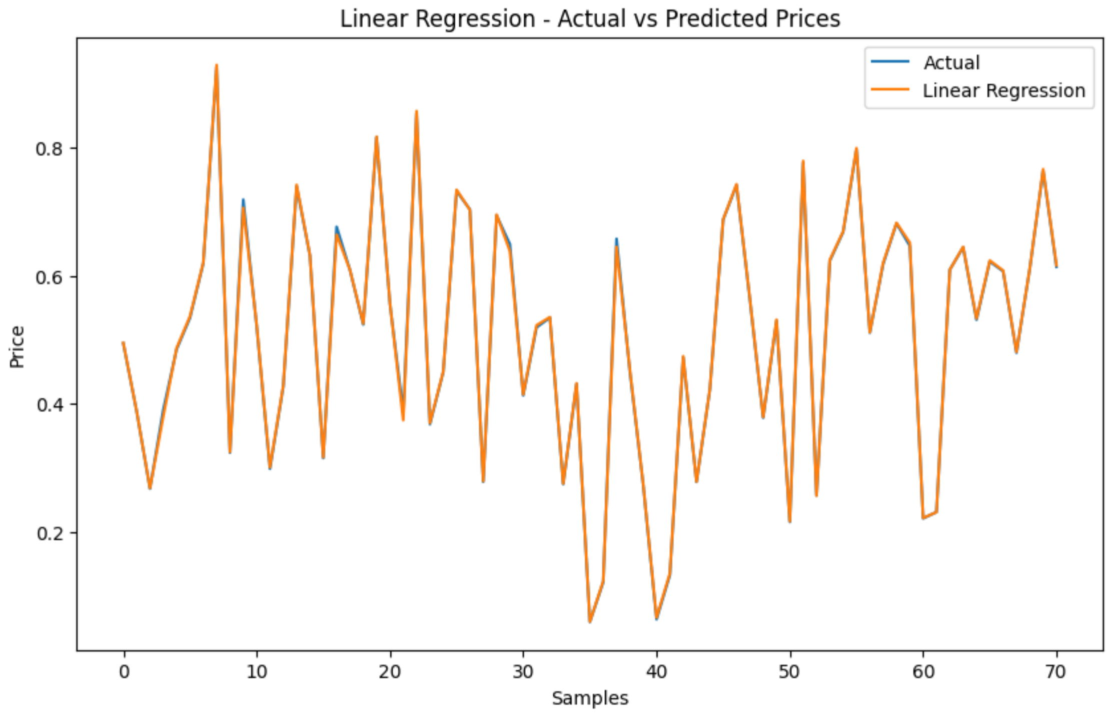
```
svr_model = SVR()
svr_model.fit(X_train_scaled, y_train)
svr_predictions = svr_model.predict(X_test_scaled)
svr_rmse = np.sqrt(mean_squared_error(y_test, svr_predictions))
svr_mae = mean_absolute_error(y_test, svr_predictions)

svr_accuracy = svr_model.score(X_test_scaled, y_test)
```

```
plt.figure(figsize=(10, 6))
plt.plot(y_test.values, label='Actual')
plt.plot(svr_predictions, label='SVR')
plt.title('SVR - Actual vs Predicted Prices')
plt.xlabel('Samples')
plt.ylabel('Price')
plt.legend()
plt.show()
```



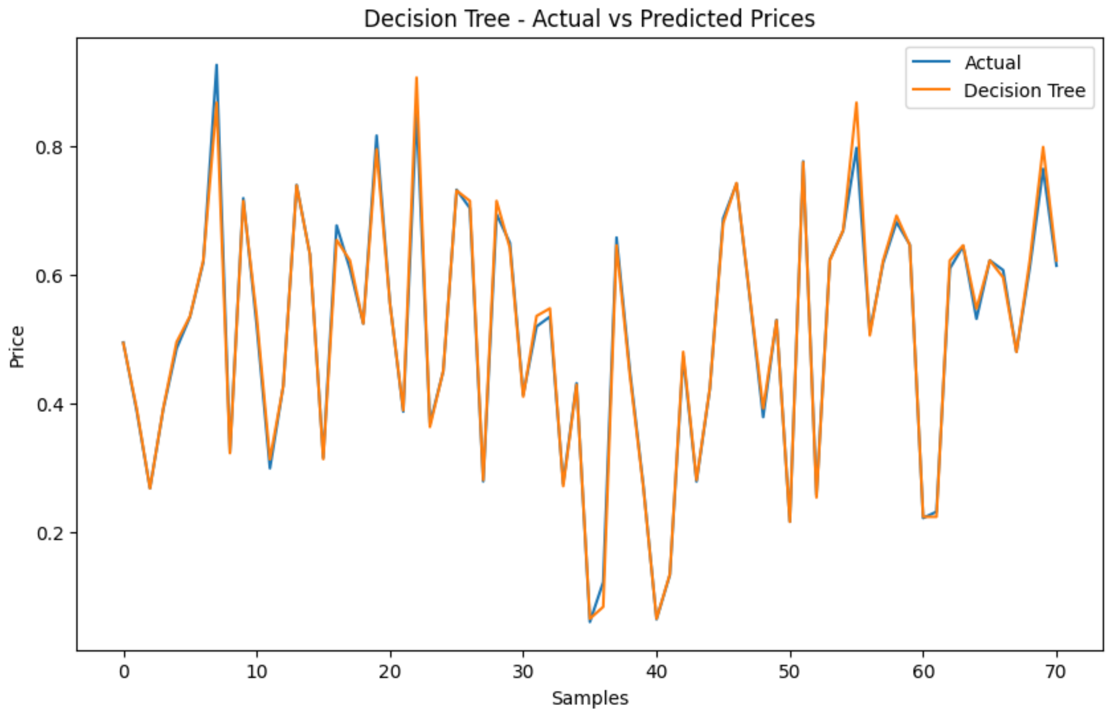
Linear Regression model

```
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)
linear_predictions = linear_model.predict(X_test_scaled)
linear_rmse = np.sqrt(mean_squared_error(y_test, linear_predictions))
linear_mae = mean_absolute_error(y_test, linear_predictions)

linear_accuracy = linear_model.score(X_test_scaled, y_test)

plt.figure(figsize=(10, 6))
plt.plot(y_test.values, label='Actual')
plt.plot(linear_predictions, label='Linear Regression')
plt.title('Linear Regression - Actual vs Predicted Prices')
plt.xlabel('Samples')
plt.ylabel('Price')
plt.legend()
plt.show()
```



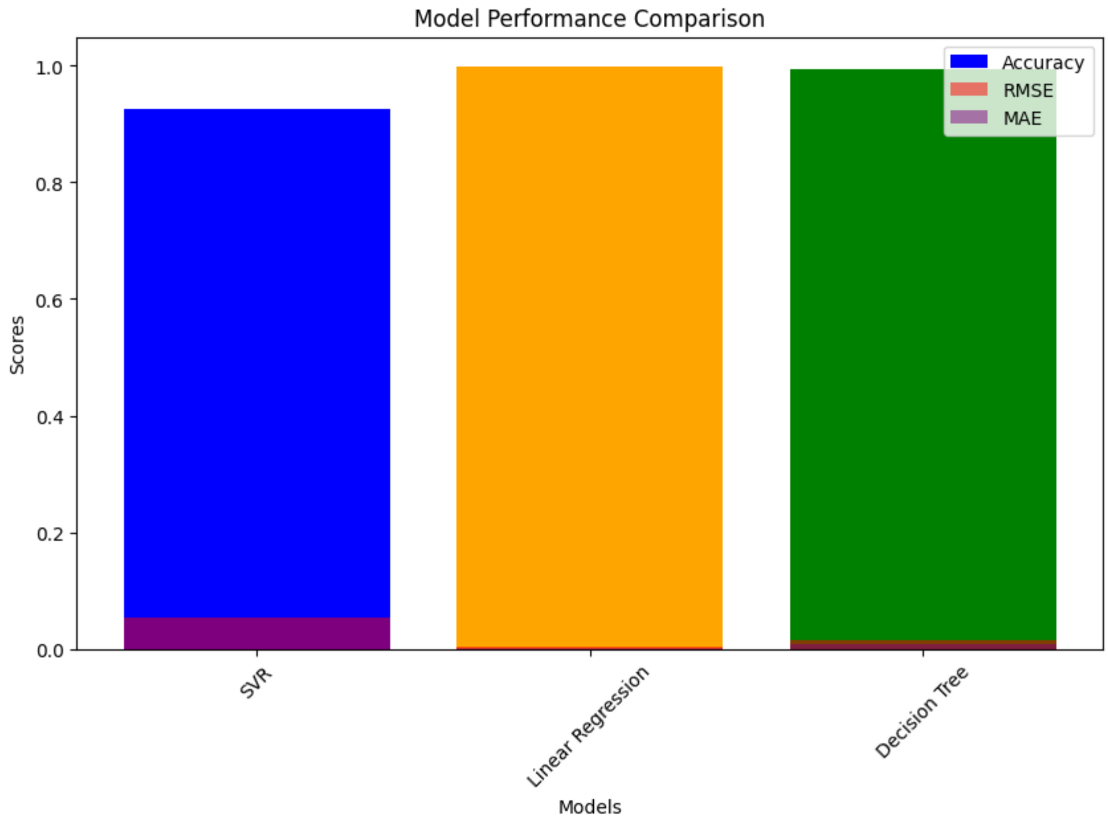
Decision Tree model

```
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train_scaled, y_train)
dt_predictions = dt_model.predict(X_test_scaled)
dt_rmse = np.sqrt(mean_squared_error(y_test, dt_predictions))
dt_mae = mean_absolute_error(y_test, dt_predictions)

dt_accuracy = dt_model.score(X_test_scaled, y_test)

plt.figure(figsize=(10, 6))
plt.plot(y_test.values, label='Actual')
plt.plot(dt_predictions, label='Decision Tree')
plt.title('Decision Tree - Actual vs Predicted Prices')
plt.xlabel('Samples')
plt.ylabel('Price')
plt.legend()
plt.show()
```

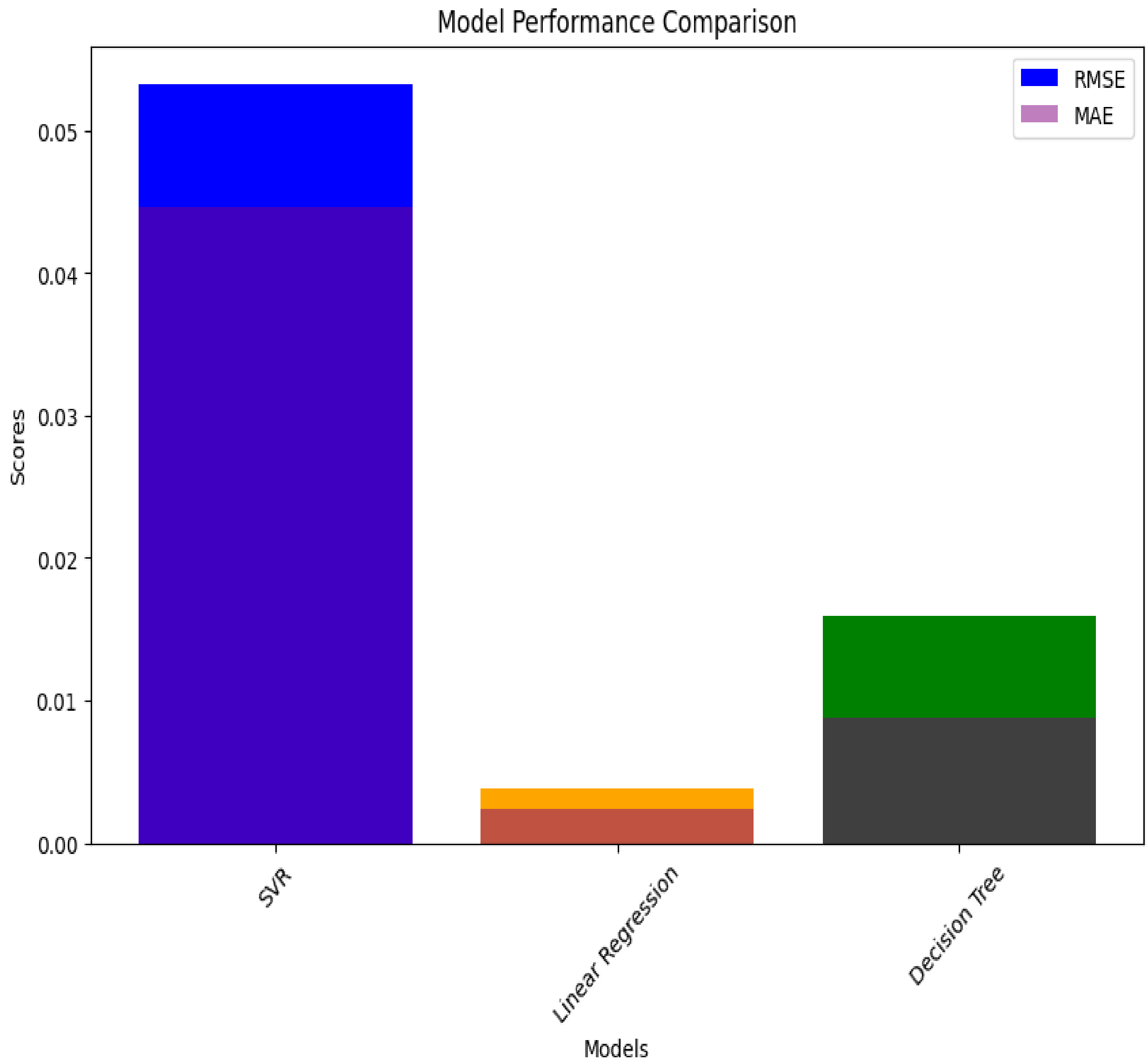


Create a bar graph to compare the accuracies, RMSE, and MAE

```
models = ['SVR', 'Linear Regression', 'Decision Tree']
accuracies = [svr_accuracy, linear_accuracy, dt_accuracy]
rmse_scores = [svr_rmse, linear_rmse, dt_rmse]
mae_scores = [svr_mae, linear_mae, dt_mae]

plt.figure(figsize=(10, 6))
colors = ['blue', 'orange', 'green']
plt.bar(models, accuracies, color=colors, label='Accuracy')
plt.bar(models, rmse_scores, color='red', alpha=0.5, label='RMSE')
plt.bar(models, mae_scores, color='purple', alpha=0.5, label='MAE')
plt.title('Model Performance Comparison')
plt.xlabel('Models')
plt.ylabel('Scores')
plt.legend()
plt.xticks(rotation=45)

plt.show()
```




```
# Create a bar graph to compare the RMSE and MAE
models = ['SVR', 'Linear Regression', 'Decision Tree']
rmse_scores = [svr_rmse, linear_rmse, dt_rmse]
mae_scores = [svr_mae, linear_mae, dt_mae]

plt.figure(figsize=(10, 6))
colors = ['blue', 'orange', 'green']
plt.bar(models, rmse_scores, color=colors, label='RMSE')
plt.bar(models, mae_scores, color='purple', alpha=0.5, label='MAE')
plt.title('Model Performance Comparison')
plt.xlabel('Models')
plt.ylabel('Scores')
plt.legend()
plt.xticks(rotation=45)

plt.show()
```


Model Performance Comparison
RMSE
MAE
0.05
0.04
0.03
0.02
0.01
0.00
Scores
SVR
Linear Regression
Decision Tree
Models

FUTURE SCOPE OF IMPROVEMENTS

- More Features: Explore adding more factors or variables that might affect stock prices, such as technical indicators, financial ratios, or market sentiment data.
- Try Different Models: Experiment with different machine learning algorithms to see which one performs best for the task. Consider using ensemble methods that combine the predictions of multiple models.
- Fine-tune Parameters: Adjust the settings of the models to find the best combination of values that improve prediction accuracy.
- Real-time Updates: Extend the project to make predictions in real-time and update the models regularly with the latest data.
- Explainability and Interpretability: Enhancing the interpretability of stock prediction models is crucial for building trust and understanding. Developing techniques to explain the model's decision-making process and the underlying factors driving the predictions will be valuable.

Certificate

This is to certify that Mr. Kovidsai Vemuri of Lovely Professional University, registration number: 12114491, has successfully completed a project on *Visualising and Forecasting Stocks* by using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Globsyn Finishing School

Certificate

This is to certify that Mr. Pawan Kumar of Lovely Professional University, registration number: 12100975, has successfully completed a project on *Visualising and Forecasting Stocks* by using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Globsyn Finishing School