Artificial Intelligence

LAB PROJECT SUBMISSION

Submitted to: Ms Swati Kumari Submitted by:
Aaditya Vardhan (102117021), Shubham Gandhi (102117007),
Rahul Divi (102117009), Aryan Raghuwanshi (102117030)

GitHub Repo: *https://github.com/intrinsicvardhan/EDAonAdaniEnt.git*

Problem Statement

Predict the Adani Enterprises Stock Price for the next 30 days.

There are Open, High, Low and Close prices that you need to obtain from the web for each day starting from 2015 to 2022 for Adani Enterprises stock.

- Split the last year into a test set- to build a model to predict stock price.
- Find short term & long-term trends.
- Understand how it is impacted from external factors or any big external events.
- Forecast for next 30 days.

Description of problem:

- The project finds Open, High, Low and Close prices
- Understanding of the external and internal factors of the stock
- Forecast for the next 30 days

Collection of Dataset

- For this project, we will be using the Yfinance library to get the data, which makes it easy to process.
- We collected data from 1-Jan-2015 to 28-Feb-2023.
- You can also download data from 'Yahoo! Finance' website. You can use Below link.
- https://finance.yahoo.com/quote/adanient.NS/history?p=adanient.NS

About the data

- Date: Date of trade

- Open: Opening Price of Stock
- High: Highest price of stock on that day
- Low: Lowest price of stock on that day
- Close: Close price adjusted for splits.
- Adj Close: Adjusted close price adjusted for splits and dividend and/or capital gain distributions.
- Volume: Volume of stock on that day

Brief description of the imported modules:

1)Pandas: - Importing pandas means bringing all of the panda's functionality to your fingertips in your Python script or Jupyter notebook
2)NumPy: -: NumPy contains a multi-dimensional array and matrix data structures. It can be utilized to perform a number of mathematical operations on arrays

3) matplotlib: Matplotlib is a multi-platform data visualization library built on NumPy

arrays. It allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram, etc

4)Seaborn: - Seaborn is a library mostly used for statistical plotting in Python. It is built on top of Matplotlib and provides beautiful default styles and color palettes to make statistical plots more attractive.

5)yfinance: - The yfinance is one of the famous modules in Python, which is used to collect online data, and with it, we can collect the financial data of Yahoo. With the help of the yfinance module, we retrieve and collect the company's financial information (such as financial ratios, etc.) as well as the histories of marketing data by using its functions

6)Warning: - The warnings module was introduced in PEP 230 as a way to warn programmers about changes in language or library features in anticipation of backwards inc

EDA

Analysis is only based on Open, High, Low, close price and volume

```python
# Importing dataset
data=yf.download('adanient.NS', start='2015-1-1', end='2023-2-28').reset_index(drop=False)
adani_0 = pd.DataFrame(data)
```

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

Using the yf library to download stock price data for the Adani Enterprises stock listed. Code downloads the stock price data for Adani Enterprises from January 1, 2015, to February 28, 2023, and stores it in a Pandas DataFrame called adani_0.

The reset_index(drop=False) method resets the index of the DataFrame so that the date column becomes a regular column in the DataFrame. The drop=False argument ensures that the original index column is retained as a regular column in the DataFrame.

```python
# Removing "Adj Close" columnfrom dataset
adani_1=adani_0.drop(["Adj Close"],axis=1).reset_index(drop=True)
adani_1
```

The resulting DataFrame is then assigned to a new variable called "adani_1". The "reset_index(drop=True)" method call ensures that the new DataFrame's index is reset and that the old index is dropped. This is done to ensure that the index is continuous, starting from 0, and that it is aligned with the new DataFrame's row count.

ompatible changes coming with Python 3.0

```python
# Finding duplicate columns, if any
adani_1[adani_1.duplicated()]
```

```python
# Finding null values, if any
adani_1.isnull().sum()
```

```python
adani_1[adani_1.isnull().any(axis=1)]
```

"adani_1[adani_1.isnull().any(axis=1)]" selects all the rows from the "adani_1" DataFrame that contain at least one missing value in any of their columns, and returns a new DataFrame containing only those rows. This can be useful for identifying and handling missing data in financial datasets

```
adani_2=adani_1.dropna().reset_index(drop=True)
adani_2
```

droping all rows containing missing values (i.e., null values) from the "adani_1" DataFrame using the "dropna()" method of the DataFrame. This method removes any row that contains at least one null value in any of its columns.

```
# Checking wether if there exist any null values
adani_2[adani_2.isnull().any(axis=1)]
```

```
# Making a copy of dataset as adani
adani=adani_2.copy()
adani
```

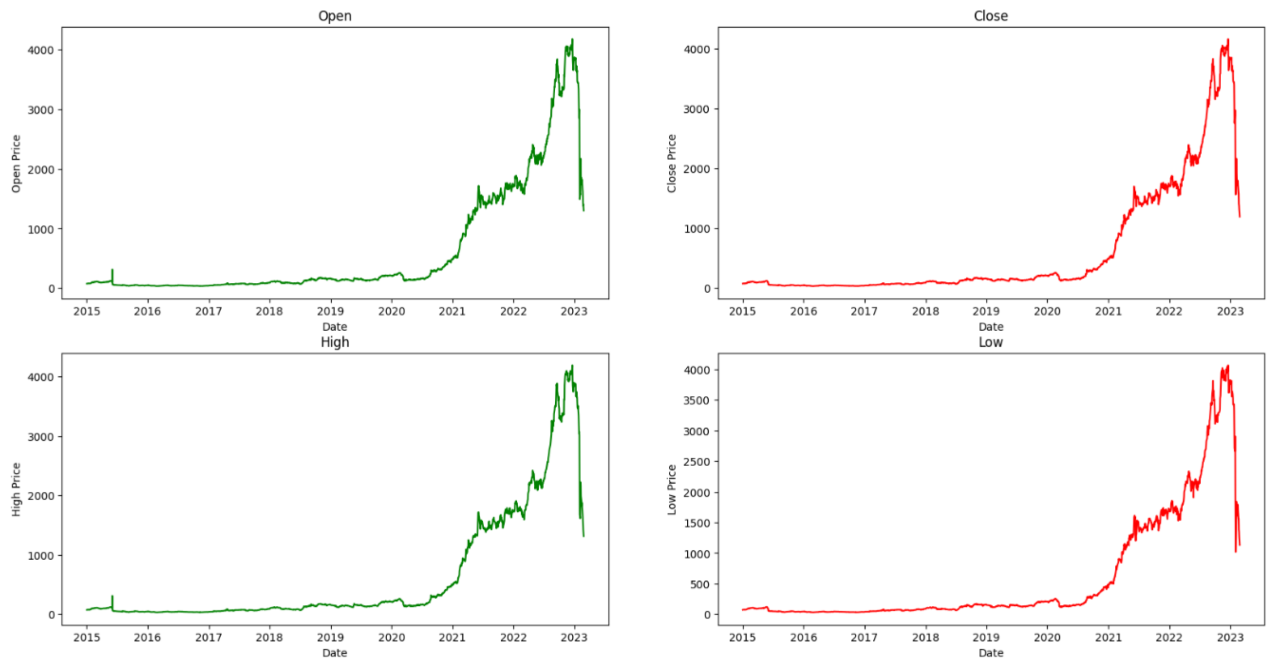
Descriptive Statistics

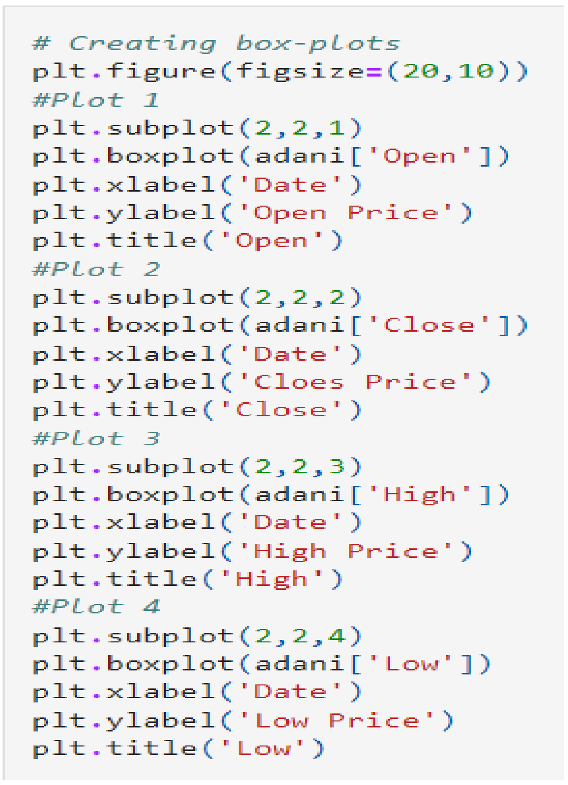
1)**adani.info()** :-In general, the info() method provides a concise summary of the DataFrame's metadata.
2) **adani.describe()** :-In general, the describe() method provides a statistical summary of the numerical columns in a Pandas DataFrame.
3) **adani.corr()** :- By examining the output of adani.corr(), one can gain insights into the relationships between the different variables in the DataFrame

Visualizations The following code is using the matplotlib.pyplot library to create four plots of the Open, Close, High and Low prices of Adani Enterprises**.**

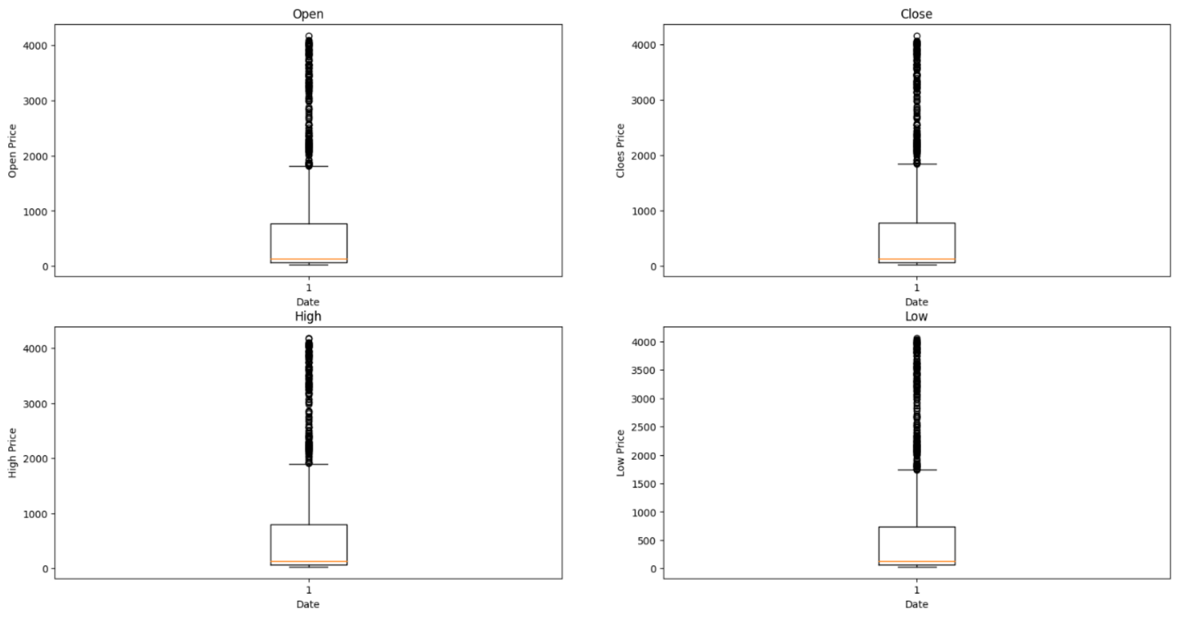
```
plt.figure(figsize=(20,10))
#Plot 1
plt.subplot(2,2,1)
plt.plot(adani['Open'],color='green')
plt.xlabel('Date')
plt.ylabel('Open Price')
plt.title('Open')
#Plot 2
plt.subplot(2,2,2)
plt.plot(adani['Close'],color='red')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Close')
#Plot 3
plt.subplot(2,2,3)
plt.plot(adani['High'],color='green')
plt.xlabel('Date')
plt.ylabel('High Price')
plt.title('High')
#Plot 4
plt.subplot(2,2,4)
plt.plot(adani['Low'],color='red')
plt.xlabel('Date')
plt.ylabel('Low Price')
plt.title('Low')
```

```
Text(0.5, 1.0, 'Low')
```



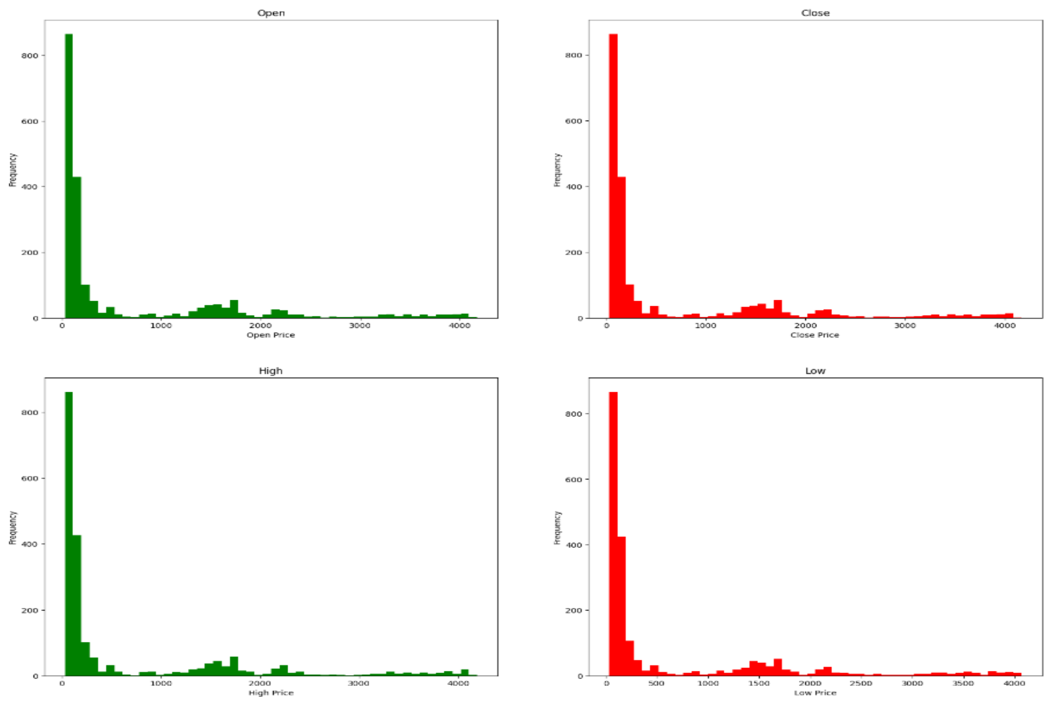


```
# Creating box-plots
plt.figure(figsize=(20,10))
#Plot 1
plt.subplot(2,2,1)
plt.boxplot(adani['Open'])
plt.xlabel('Date')
plt.ylabel('Open Price')
plt.title('Open')
#Plot 2
plt.subplot(2,2,2)
plt.boxplot(adani['Close'])
plt.xlabel('Date')
plt.ylabel('Cloes Price')
plt.title('Close')
#Plot 3
plt.subplot(2,2,3)
plt.boxplot(adani['High'])
plt.xlabel('Date')
plt.ylabel('High Price')
plt.title('High')
#Plot 4
plt.subplot(2,2,4)
plt.boxplot(adani['Low'])
plt.xlabel('Date')
plt.ylabel('Low Price')
plt.title('Low')
```



The term "box plot" comes from the fact that the graph looks like a rectangle with lines extending from the top and bottom. Because of the extending lines, this type of graph is sometimes called a box-and-whisker plot. The distances between the different box parts represent the degree of a data dispersion and a data asymmetry to identify outliers.

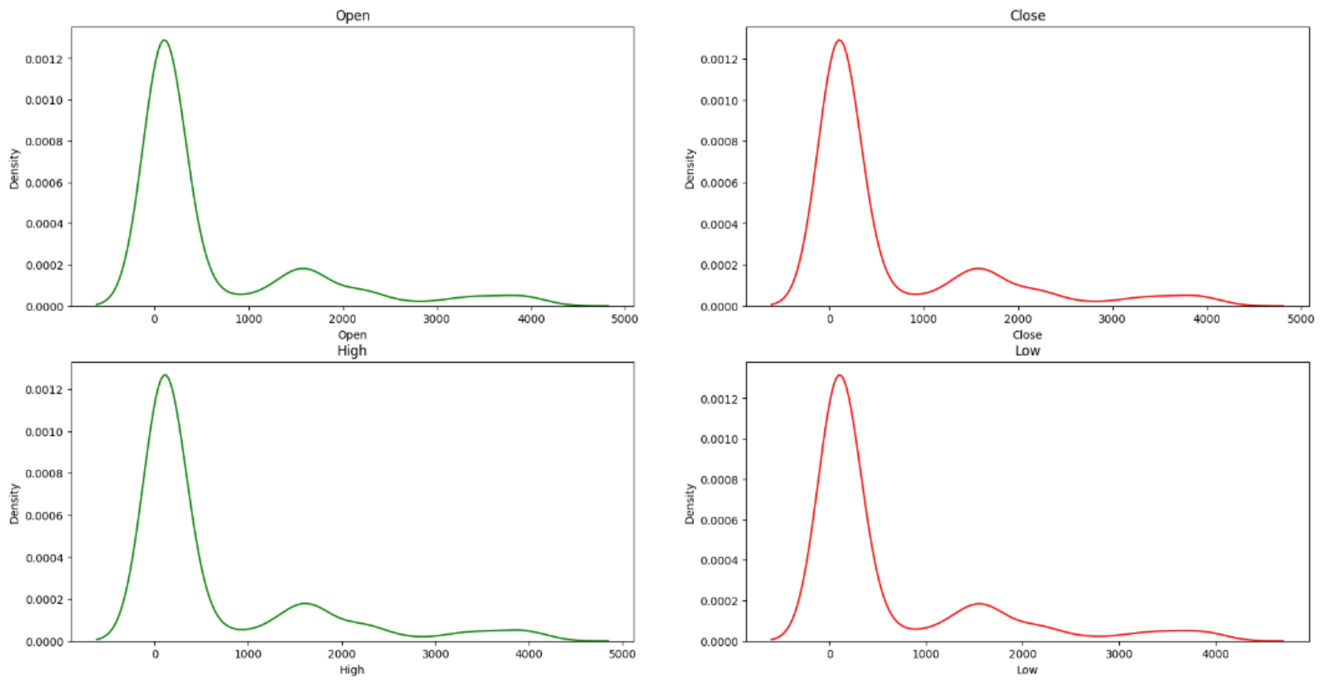
```
# Ploting Histogram
plt.figure(figsize=(20,18))
#Plot 1
plt.subplot(2,2,1)
plt.hist(adani['Open'],bins=50, color='green')
plt.xlabel("Open Price")
plt.ylabel("Frequency")
plt.title('Open')
#Plot 2
plt.subplot(2,2,2)
plt.hist(adani['Close'],bins=50, color='red')
plt.xlabel("Close Price")
plt.ylabel("Frequency")
plt.title('Close')
#Plot 3
plt.subplot(2,2,3)
plt.hist(adani['High'],bins=50, color='green')
plt.xlabel("High Price")
plt.ylabel("Frequency")
plt.title('High')
#Plot 4
plt.subplot(2,2,4)
plt.hist(adani['Low'],bins=50, color='red')
plt.xlabel("Low Price")
plt.ylabel("Frequency")
plt.title('Low')
```



A histogram is a bar graph-like representation of data that buckets a range of classes into columns along the horizontal x-axis.In trading, the MACD histogram is used by technical analysts to indicate changes in momentum.

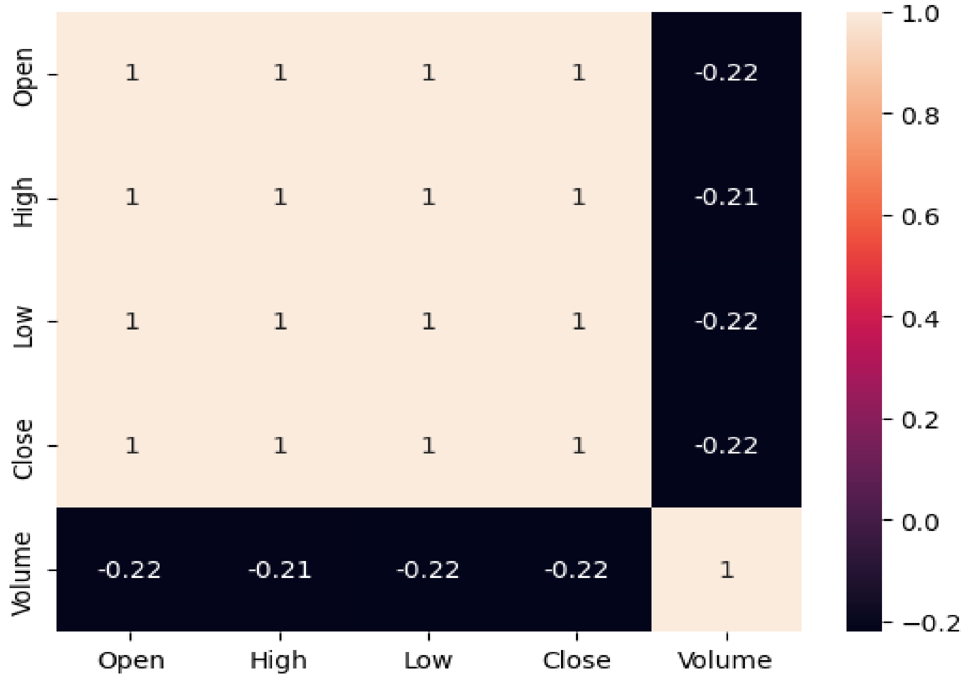
```
# KDE-Plots
plt.figure(figsize=(20,10))
#Plot 1
plt.subplot(2,2,1)
sns.kdeplot(adani['Open'], color='green')
plt.title('Open')
#Plot 2
plt.subplot(2,2,2)
sns.kdeplot(adani['Close'], color='red')
plt.title('Close')
#Plot 3
plt.subplot(2,2,3)
sns.kdeplot(adani['High'], color='green')
plt.title('High')
#Plot 4
plt.subplot(2,2,4)
sns.kdeplot(adani['Low'], color='red')
plt.title('Low')
```

```
Text(0.5, 1.0, 'Low')
```



In a KDE plot, the distribution is estimated by placing a kernel function at each data point and summing the contributions of all kernels to obtain the overall density function. The result is a continuous curve that approximates the true underlying distribution.

```
sns.heatmap(adani.corr(),annot=True)
plt.show()
```



Heatmaps can be useful for quickly identifying trends and patterns in the performance of a group of securities. They can be particularly useful for portfolio managers who want to identify which stocks are performing well and which are underperforming in relation to their peers.

Model Building

It is a process of creating a mathematical representation , or model of a system or process using machine learning algorithms.

The process of model building typically involves several steps, including data preprocessing, selecting appropriate features , choosing an appropriate algorithm or model architecture, training model on a labeled dataset, evaluating the model’s performance on a validation set, and fine-tuning the model to improve its performance.

We take the following input:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score, r2_score
from sklearn.metrics import mean_poisson_deviance, mean_gamma_deviance, accuracy_score
from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.layers import LSTM, GRU

from itertools import cycle

import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
```

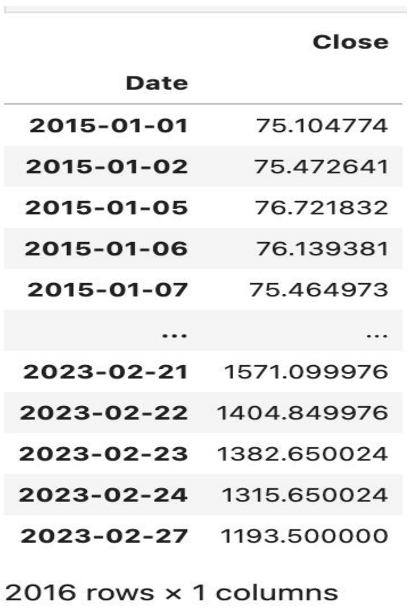
```
adani
```

	Open	High	Low	Close	Volume
Date					
2015-01-01	74.399712	75.495628	73.663994	75.104774	3946806
2015-01-02	75.304039	76.177704	75.104774	75.472641	6565229
2015-01-05	75.273384	77.641479	75.212074	76.721832	9404837
2015-01-06	75.963120	79.381149	74.215782	76.139381	18412441
2015-01-07	76.637527	77.794754	73.878578	75.464973	10863352
...	...	...	...	...	...
2023-02-21	1626.000000	1644.449951	1561.300049	1571.099976	5571915
2023-02-22	1535.000000	1560.000000	1381.199951	1404.849976	10606476
2023-02-23	1380.000000	1438.000000	1350.000000	1382.650024	8907540
2023-02-24	1410.000000	1427.000000	1261.599976	1315.650024	8736727
2023-02-27	1300.000000	1313.800049	1131.050049	1193.500000	10271008

2016 rows × 5 columns

// Creating Dataframe which only includes date and close time:

```
close_df=pd.DataFrame(adani['Close'])
close_df
```



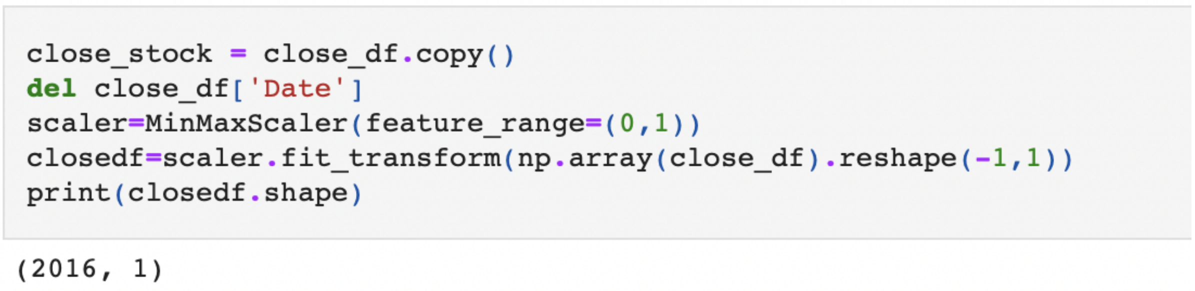
Date	Close
2015-01-01	75.104774
2015-01-02	75.472641
2015-01-05	76.721832
2015-01-06	76.139381
2015-01-07	75.464973
...	...
2023-02-21	1571.099976
2023-02-22	1404.849976
2023-02-23	1382.650024
2023-02-24	1315.650024
2023-02-27	1193.500000

2016 rows × 1 columns

Normalizing/Scaling close value between 0 to 1:

This means that youre transforming your data so that it fits within a specific scale , like 0-1 in this case. You want to scale data when youre using methods based on measures of how far apart data points are, like SVM or KNN.

Scaling just changes the range of your data.Normalisation is more radical transformation. The point of normalization is to change your observations so that they can be described as a normal distribution.



```
close_stock = close_df.copy()
del close_df['Date']
scaler=MinMaxScaler(feature_range=(0,1))
closedf=scaler.fit_transform(np.array(close_df).reshape(-1,1))
print(closedf.shape)
```

```
(2016, 1)
```

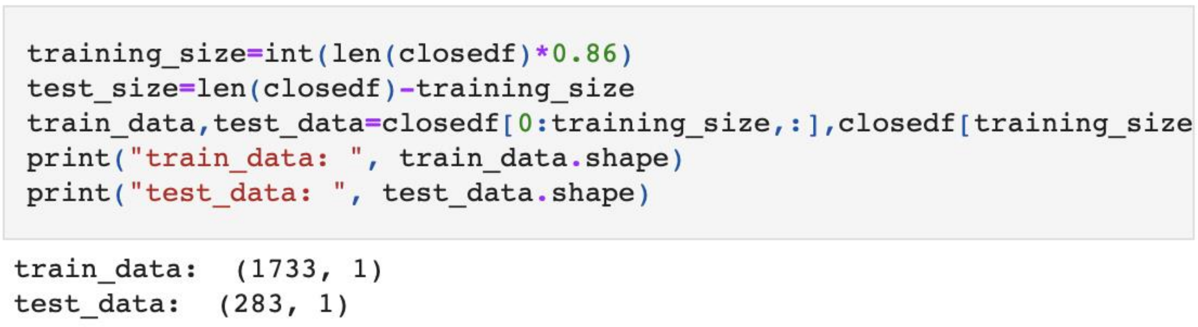
Split Data for training and testing:

The train-test split is used to estimate the performance of machine learning algorithms that are applicable for prediction-based algorithms.This method is a fast and easy procedure to perform such that we can compare our own machine learning model results to machine results.

By default, the test set is split into 30% of actual data and the training set is split into 70% of the actual data.

Scikit_learn alias sklearn is the most useful and robust library for machine learning in python. The sklearn library provides us with the model_selection module in which we have the splitter function train_test_split().

// Ratio for training and testing data is 86:14

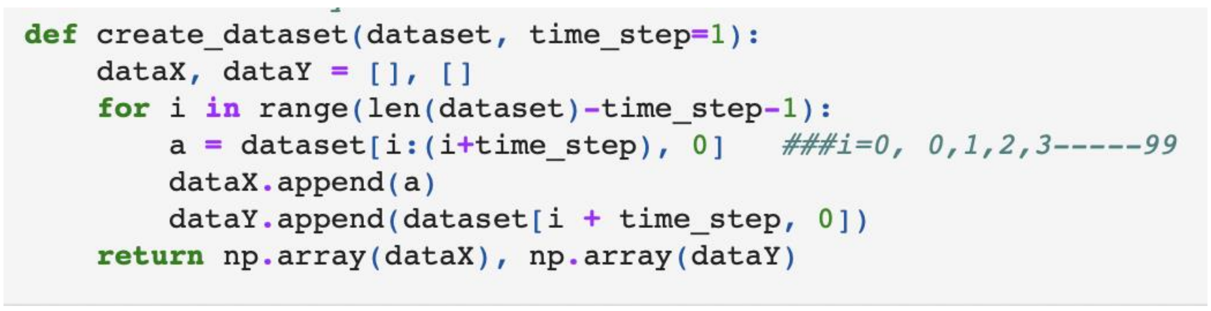


```
training_size=int(len(closedf)*0.86)
test_size=len(closedf)-training_size
train_data,test_data=closedf[0:training_size,:],closedf[training_size
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)
```

```
train_data:  (1733, 1)
test_data:  (283, 1)
```

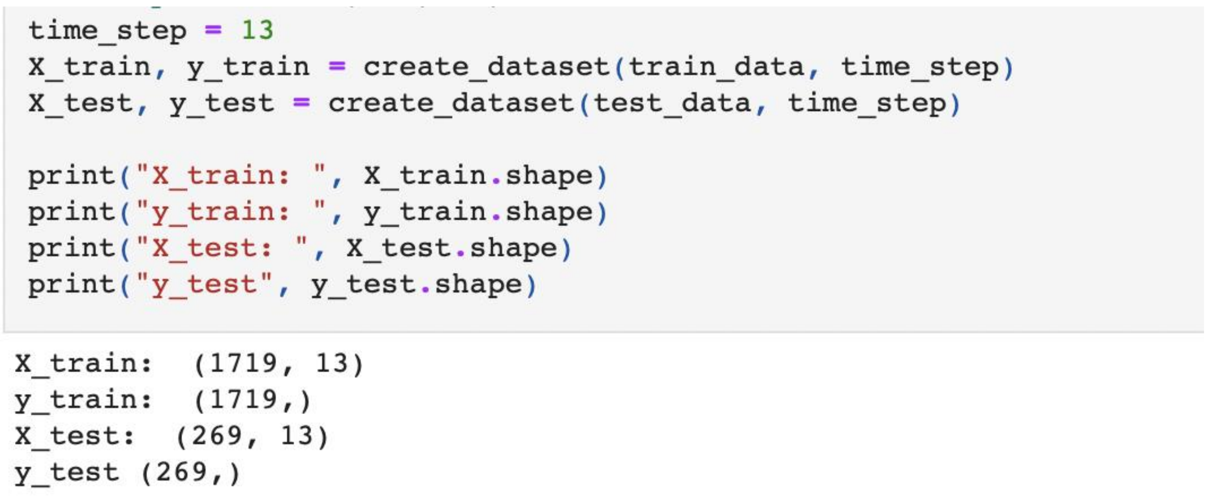
Creating new dataset according to requirement of time-series prediction:

1) **Converting an array of values into a dataset matrix:**



```
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

2) **Reshaping into X=t,t+1,t+2,t+3 and Y=t+4:**



```
time_step = 13
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train:  (1719, 13)
y_train:  (1719,)
X_test:  (269, 13)
y_test (269,)
```

Evaluation metrices RMSE, MSE and MAE:

Root Mean Square Error (RMSE), Mean Square Error (MSE) and Mean absolute Error (MAE) are a standard way to measure the error of a model in predicting quantitative data.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

MAE: The MAE is simply defined as the sum of all the distances/residual s(the difference between the actual and predicted value) divided by the total number of points in the dataset.

It is the absolute average distance of our model prediction.

RMSE: The root mean squared error, which is the square root of the average squared distance (difference between actual and predicted value).

R2 score for regression:

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that is explained by an independent variable or variables in a regression model.

1 = Best

0 or < 0 = worse

Formula

$$R^2 = 1 - \frac{RSS}{TSS}$$

R^2 = coefficient of determination
RSS = sum of squares of residuals
TSS = total sum of squares

$$\text{RSS} = \Sigma\left(y_i - \widehat{y}_i\right)^2$$

Where: y_i is the actual value and, $\widehat{y}_i$ is the predicted value.

$$\text{TSS} = \Sigma\left(y_i - \overline{y}\right)^2$$

Where: y_i is the actual value and $\overline{y}$ is the mean value of the variable/feature

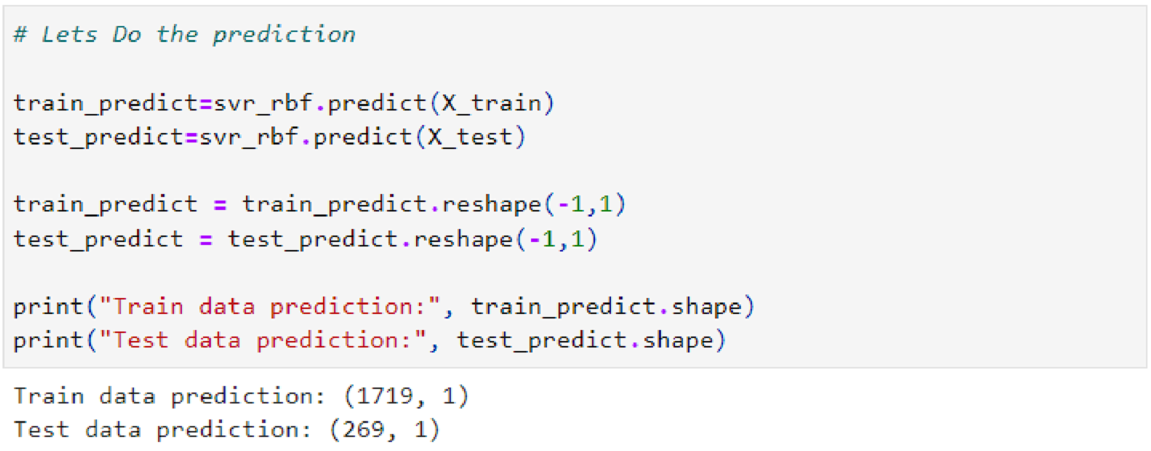
Algorithms

Support vector regression - SVR

- Support Vector Regression (SVR) is a machine learning algorithm that is used for regression problems, which involves predicting a continuous output variable.
- In simple terms, SVR works by creating a line (or hyperplane) that best fits the training data, while maximizing the margin between the line and the data points. The points closest to the line are called support vectors, and they are used to define the optimal line.
- The cost function for Support Vector Regression (SVR) is typically based on the mean squared error (MSE) between the predicted output and the actual output of the training data. The goal of SVR is to find the linear function that maximizes the margin around the support vectors while keeping the error within a certain threshold, known as the epsilon-insensitive zone.

```python
import math
from sklearn.svm import SVR

svr_rbf = SVR(kernel= 'rbf', C= 1e3, gamma= 0.01)
svr_rbf.fit(X_train, y_train)
```

```
▾ SVR
SVR(C=1000.0, gamma=0.01)
```

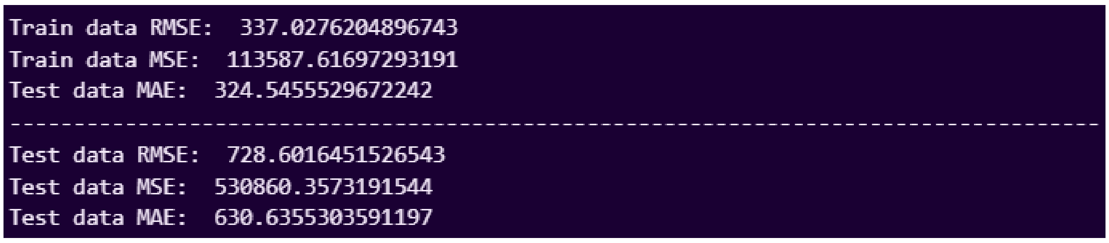


```
# Lets Do the prediction

train_predict=svr_rbf.predict(X_train)
test_predict=svr_rbf.predict(X_test)

train_predict = train_predict.reshape(-1,1)
test_predict = test_predict.reshape(-1,1)

print("Train data prediction:", train_predict.shape)
print("Test data prediction:", test_predict.shape)
```

```
Train data prediction: (1719, 1)
Test data prediction: (269, 1)
```

Evaluation metrices RMSE, MSE and MAE

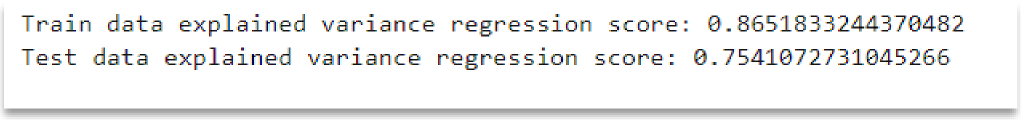
Root Mean Square Error (RMSE), Mean Square Error (MSE) and Mean absolute Error (MAE) are a standard way to measure the error of a model in predicting quantitative data.



```
Train data RMSE:  337.0276204896743
Train data MSE:  113587.61697293191
Test data MAE:  324.5455529672242
-------------------------------------------------------------------------------------
Test data RMSE:  728.6016451526543
Test data MSE:  530860.3573191544
Test data MAE:  630.6355303591197
```

Explained variance regression score

The explained variance score explains the dispersion of errors of a given dataset, and the formula is written as follows: Here, and Var(y) is the variance of prediction errors and actual values respectively. Scores close to 1.0 are highly desired, indicating better squares of standard deviations of errors.



```
Train data explained variance regression score: 0.8651833244370482
Test data explained variance regression score: 0.7541072731045266
```

R2 score for regression

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that is explained by an independent variable or variables in a regression model.

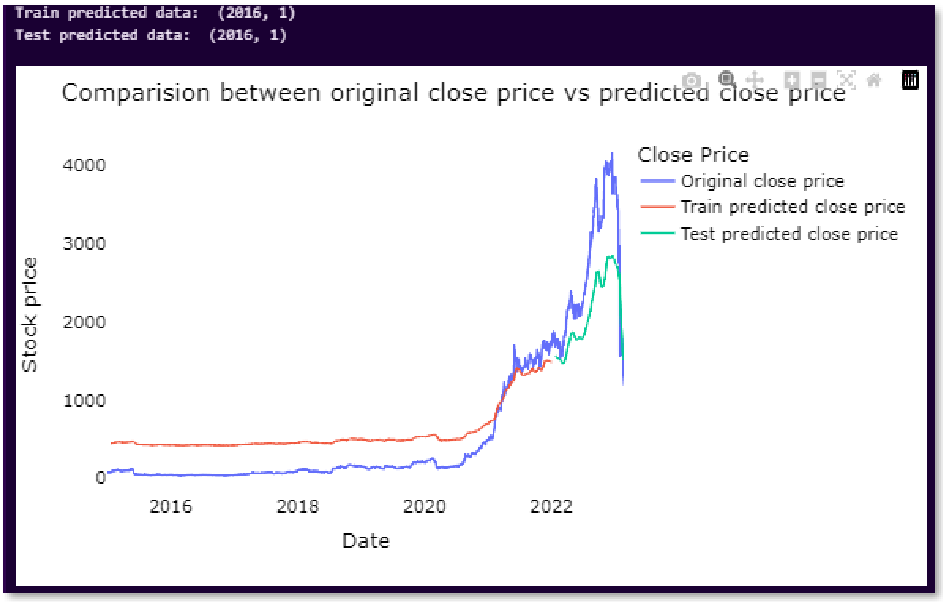
- 1 = Best
- 0 or < 0 = worse

```
train_r2_svr=r2_score(original_ytrain, train_predict)
test_r2_svr=r2_score(original_ytest, test_predict)
print("Train data R2 score:", train_r2_svr)
print("Test data R2 score:", test_r2_svr)
```

```
Train data R2 score: 0.43475359395821966
Test data R2 score: 0.22448966783520774
```

Comparison between original stock close price vs predicted close price

- Creating a plot to compare the original close price of Adani Enterprises with the predicted close price.
- Predicted values are assigned to the appropriate positions.
- Using the dataframe plotdf it is used using the function from the Plotly Express library, with the x-axis set to the date and the y-axis set to the original close price and the two predicted close prices.



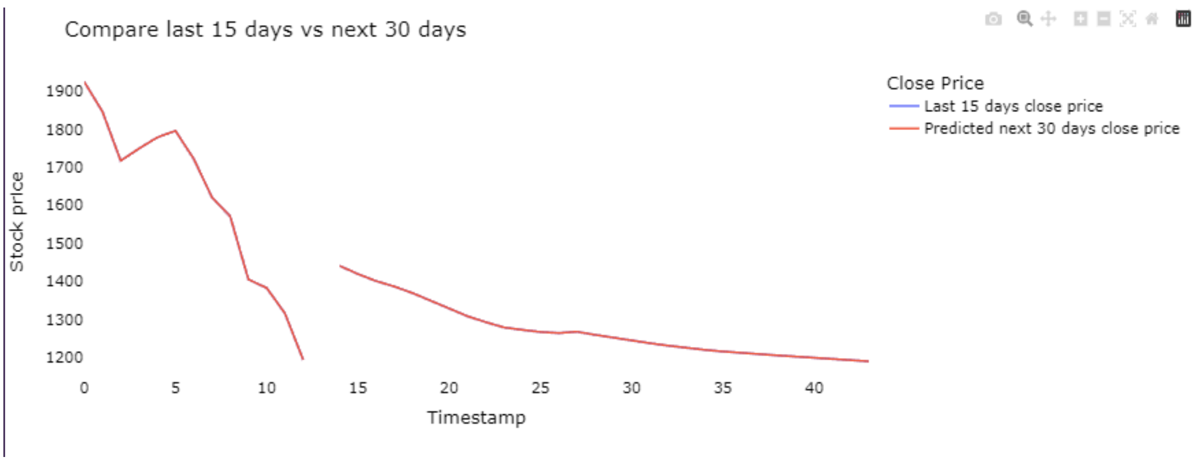
```
Train predicted data:  (2016, 1)
Test predicted data:  (2016, 1)
```

Plotting last 15 days and next predicted 30 days

- The following plot is created to compare the last 15 days of the original stock price data with the predicted next 30 days of closing stock price data using SVR.
- Plot is created using Plotly Express. The legend of the plot shows the names of the lines, “Last 15 days close price” and “Predicted next 30 days close price”.

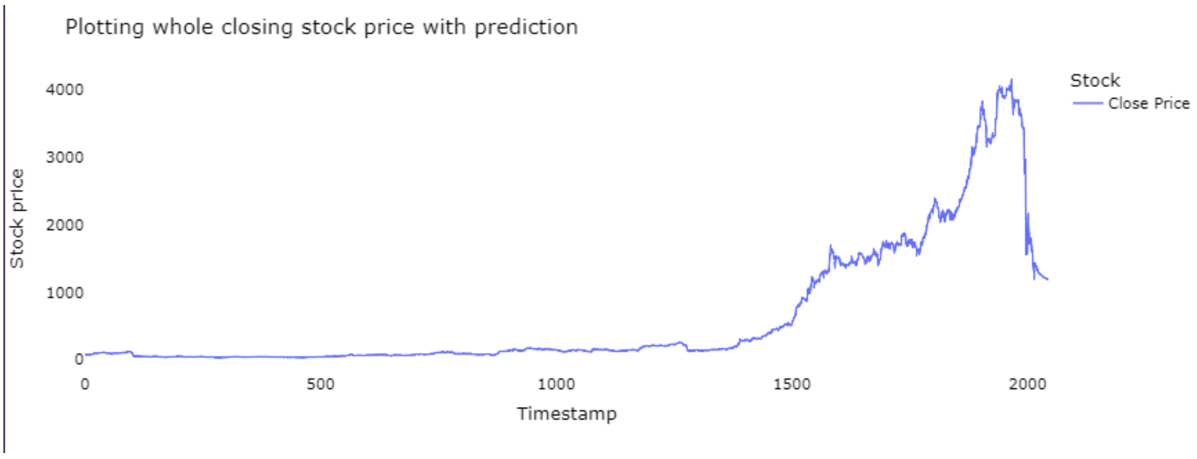
```
last_days=np.arange(1,time_step+1)
day_pred=np.arange(time_step+1,time_step+pred_days+1)
print(last_days)
print(day_pred)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13]
[14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
 38 39 40 41 42 43]
```

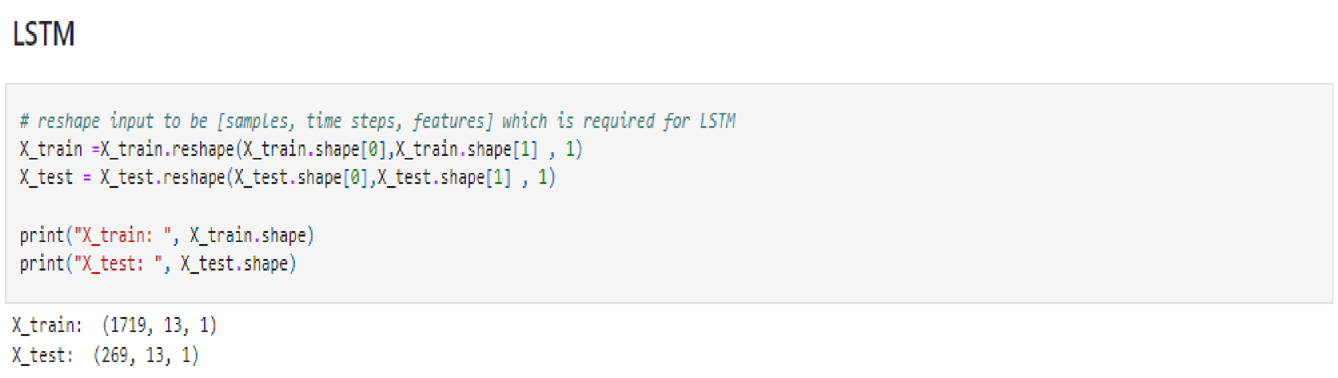


Plotting whole closing stock price with prediction

- Converting the predicted and actual stock prices from scaled form back to their original scale using the inverse_transform method of the MinMaxScaler object.
- We generate the following by using a line plot of the actual and predicted closing stock prices with some customizations using Plotly.



LSTM (Long-Short Term Memory) Algorithm



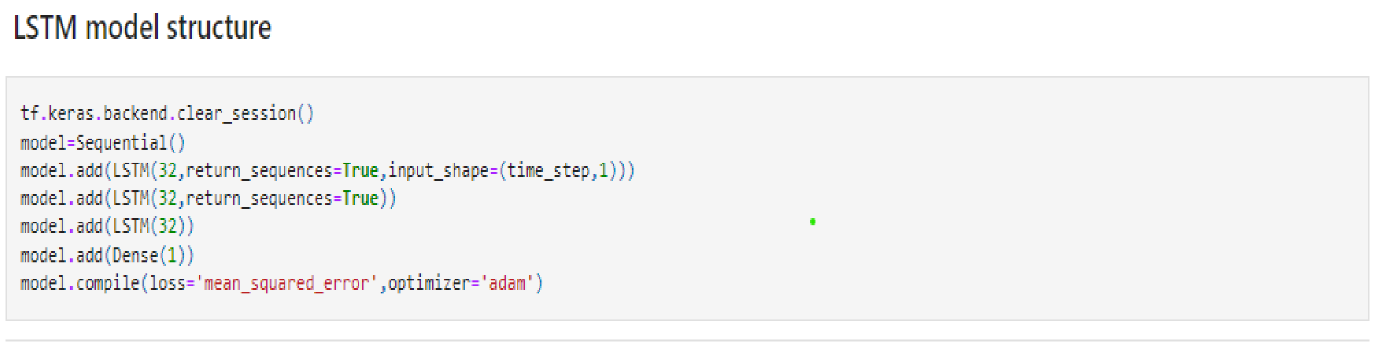
LSTM

```
# reshape input to be [samples, time steps, features] which is required for LSTM
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)

print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)
```

```
X_train:  (1719, 13, 1)
X_test:  (269, 13, 1)
```

Code is reshaping the input data. Specifically, it is reshaping the training and testing data arrays X_train and X_test from 2-dimensional arrays to 3-dimensional arrays.

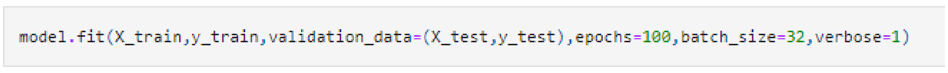
The first parameter of reshape() specifies the new number of samples, which is the same as the original number of samples. The second parameter specifies the new number of time steps, which is the same as the original number of time steps. The third parameter specifies the new number of features, which is 1 in this case.



LSTM model structure

```
tf.keras.backend.clear_session()
model=Sequential()
model.add(LSTM(32,return_sequences=True,input_shape=(time_step,1)))
model.add(LSTM(32,return_sequences=True))
model.add(LSTM(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

This initializes a new sequential model in Keras using TensorFlow backend, with three LSTM layers followed by a dense layer. The first LSTM layer has 32 units, takes in sequences of length time_step with one feature, and returns sequences. The second LSTM layer also has 32 units and returns sequences. The third LSTM layer has 32 units and does not return sequences. Finally, there is a dense layer with a single output unit. The model is compiled with mean squared error loss and the Adam optimizer.

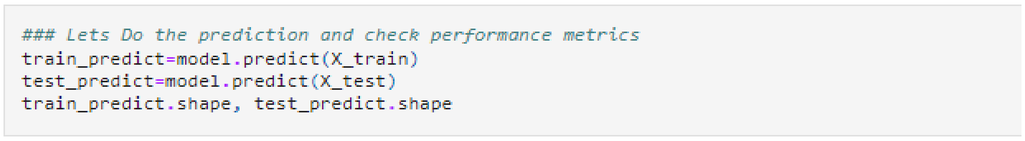


```
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100,batch_size=32,verbose=1)
```

Trains on the training data and labels, using a validation set to monitor performance, for a total of 100 epochs, with batches of 32 samples at a time, and progress updates displayed for each epoch.

The epochs parameter determines how many times the model will iterate over the entire training dataset during the training process.

The fit() method is used to train the model using the training data X_train and corresponding labels y_train. The validation_data argument is used to provide a set of validation data to evaluate the model's performance during training. Here, X_test and y_test are the validation data.

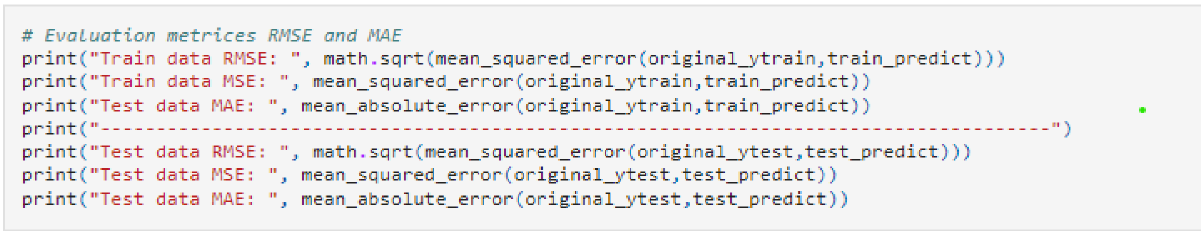


```
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
train_predict.shape, test_predict.shape
```

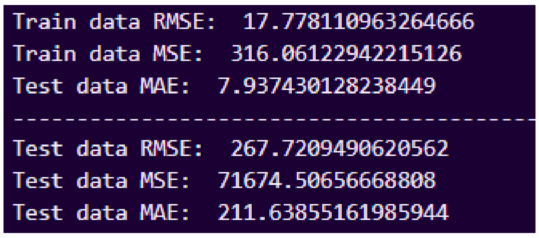
Piece of code doing the prediction and checking the performance metrics

The line train_predict=model.predict(X_train) makes predictions on the X_train data using the pre-trained model. The resulting predictions are stored in the train_predict variable.

Evaluation metrices RMSE, MSE and MAE



```
# Evaluation metrices RMSE and MAE
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_predict)))
print("Train data MSE: ", mean_squared_error(original_ytrain,train_predict))
print("Test data MAE: ", mean_absolute_error(original_ytrain,train_predict))
print("-------------------------------------------------------------------------------------")
print("Test data RMSE: ", math.sqrt(mean_squared_error(original_ytest,test_predict)))
print("Test data MSE: ", mean_squared_error(original_ytest,test_predict))
print("Test data MAE: ", mean_absolute_error(original_ytest,test_predict))
```



```
Train data RMSE:  17.778110963264666
Train data MSE:  316.06122942215126
Test data MAE:  7.937430128238449
----------------------------------------
Test data RMSE:  267.7209490620562
Test data MSE:  71674.50656668808
Test data MAE:  211.63855161985944
```

The code calculates these metrics for both the training and testing data, which allows to compare how well the model is generalizing to new data.

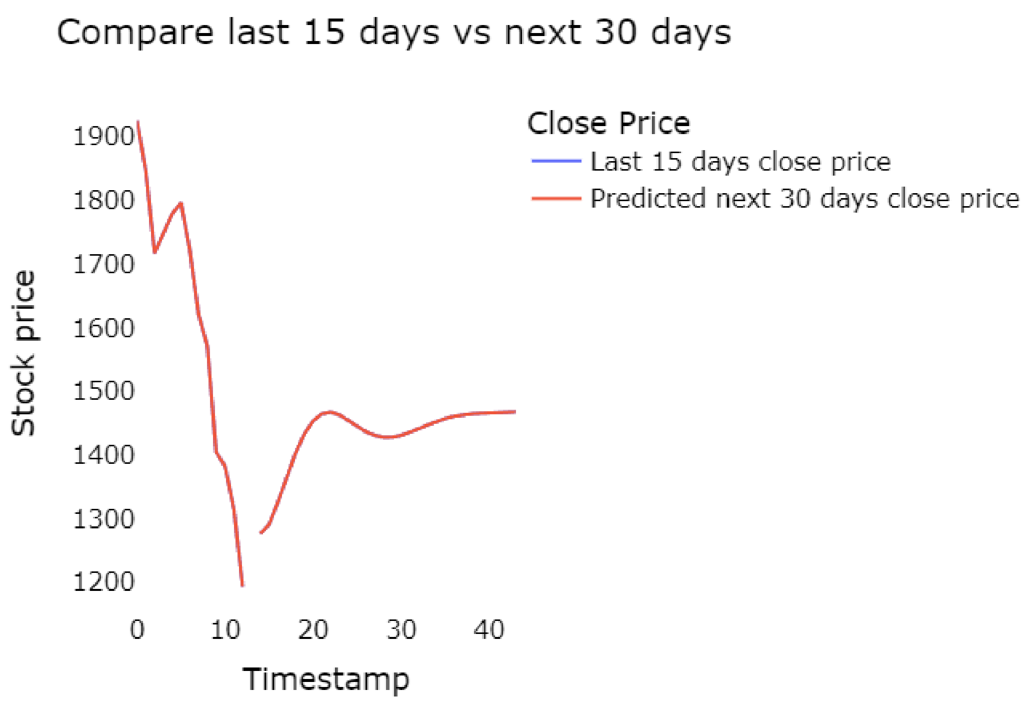
- RMSE (Root Mean Squared Error): This metric calculates the average difference between the predicted and actual values, considering the squared values of the differences. The square root of this value is taken to make the units of the metric the same as the original values. Lower values indicate better model performance.
- MSE (Mean Squared Error): This is the average of the squared differences between the predicted and actual values. Higher values indicate poorer model performance.
- MAE (Mean Absolute Error): This metric calculates the average difference between the predicted and actual values, but without taking the square of the differences. Lower values indicate better model performance.

Comparison between original stock close price vs predicted close price

The arrays are initialized with Nun values, and the predicted closing prices are then assigned to the appropriate locations in the arrays. The 'plotdf' DataFrame is then created, which contains the original closing price, the predicted closing price for the training data, and the predicted closing price for the testing data. Finally, the 'plotly' library is used to create a line plot of the data, with the original closing price, the predicted closing price for the training data, and the predicted closing price for the testing data plotted on the same graph.

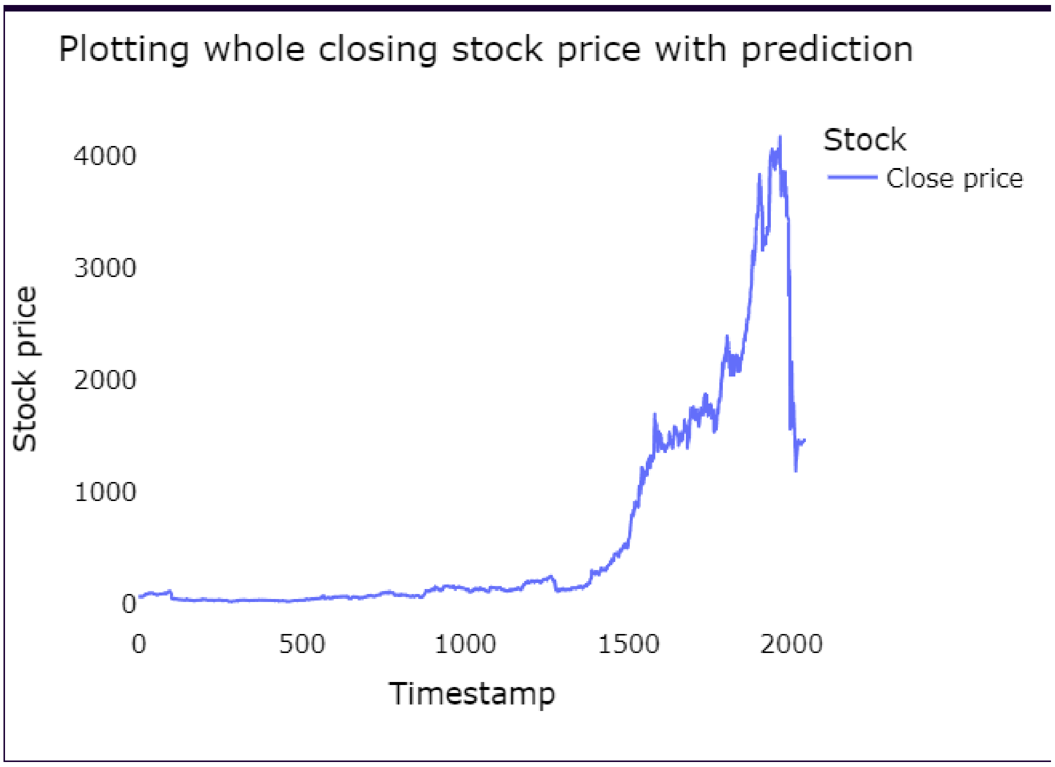
Plotting last 15 days and next predicted 30 days

```
last_days=np.arange(1,time_step+1)
day_pred=np.arange(time_step+1,time_step+pred_days+1)
print(last_days)
print(day_pred)
```



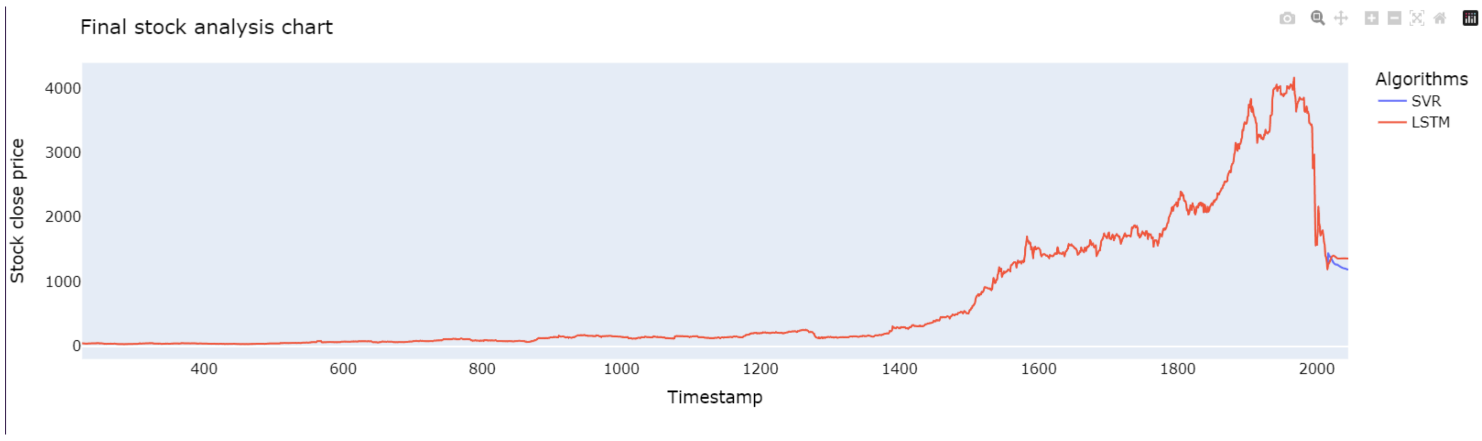
Plotting whole closing stock price with prediction

Using the 'tolist()' function to convert a Pandas DataFrame called 'closedf' to a Python list called 'lstmdf' It then uses extend() to append the values from another list called 'lst_output' to 'lstmdf'. The resulting list is then transformed using a scaler object called scaler. The 'inverse_transform()' function is used to undo any previous scaling on the data, and the resulting values are reshaped and converted back to a list.

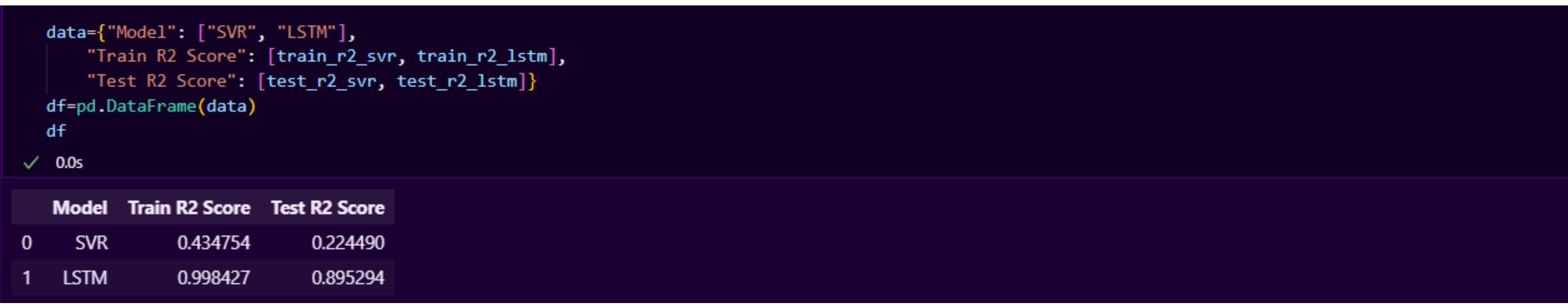


The code then creates a plot using the 'plotly.express' library. The 'px.line()' function is used to create a line plot of the data in lstmdf, with labels and a title added to the plot. The plot is then customized using various functions, such as 'update_xaxes()' and 'update_yaxes()', to remove the gridlines. Finally, the plot is displayed using 'fig.show()'.

Conclusion Chart



By looking into the following table, we can say that our LSTM model have best R2 score.



```
data={"Model": ["SVR", "LSTM"],
    "Train R2 Score": [train_r2_svr, train_r2_lstm],
    "Test R2 Score": [test_r2_svr, test_r2_lstm]}
df=pd.DataFrame(data)
df
```

✓ 0.0s

	Model	Train R2 Score	Test R2 Score
0	SVR	0.434754	0.224490
1	LSTM	0.998427	0.895294