Artificial Intelligence

LAB PROJECT SUBMISSION

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

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 Industries 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.
- But also you can download data from 'Yahoo! Finance' website. You can use Below link.
- https://finance.yahoo.com/quote/adani.NS/history?p=adani.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

Importing Used Modules

- Brief Description of the imported modules

1. *pandas* is a popular data manipulation library for Python, providing easy-to-use data structures and data analysis tools for handling large datasets.
2. *numpy* is a numerical computing library for Python, providing tools for working with arrays, matrices and numerical computations.
3. *matplotlib* is a numerical computing library for Python, providing tools for working with arrays, matrices, and numerical computations.
4. *seaborn* is a data visualisation library based on matplotlib, providing high-level interface for creaitng statistical graphics.
5. *yfinance* is a library for retrieving financial data from Yahoo Finance. It provides and easy to use interface for accessing historical and real-time financial data for various markets, including stocks, currencies, and commodities.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
# Importing dataset
data=yf.download('adanient.NS', start='2015-1-1', end='2023-2-28').reset_index(c
adani_0 = pd.DataFrame(data)
```

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

In [ ]:
```
adani_0.head(10)
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-01-01	74.399712	75.495628	73.663994	75.104774	70.727562	3946806
1	2015-01-02	75.304039	76.177704	75.104774	75.472641	71.073997	6565229
2	2015-01-05	75.273384	77.641479	75.212074	76.721832	72.250381	9404837
3	2015-01-06	75.963120	79.381149	74.215782	76.139381	71.701866	18412441
4	2015-01-07	76.637527	77.794754	73.878578	75.464973	71.066765	10863352
5	2015-01-08	76.530235	78.783379	76.453598	78.438515	73.867012	7942903
6	2015-01-09	79.089928	79.817986	76.330978	77.656807	73.130859	12083224
7	2015-01-12	78.269905	80.070892	78.170280	78.867683	74.271156	12302379
8	2015-01-13	79.388817	79.519096	75.863487	76.131721	71.694664	7711411
9	2015-01-14	76.407616	76.438271	74.491676	75.311699	70.922424	6088603

In [ ]:

```
adani_0.tail(10)
```

Out[ ]:

	Date	Open	High	Low	Close	Adj Close	Volume
2006	2023-02-14	1735.0	1889.000000	1611.349976	1749.699951	1749.699951	14579030
2007	2023-02-15	1780.0	1824.400024	1750.000000	1779.099976	1779.099976	7636578
2008	2023-02-16	1820.0	1874.949951	1790.000000	1796.599976	1796.599976	5578515
2009	2023-02-17	1800.0	1815.849976	1703.199951	1722.699951	1722.699951	5392513
2010	2023-02-20	1650.0	1685.000000	1560.500000	1621.449951	1621.449951	6762330
2011	2023-02-21	1626.0	1644.449951	1561.300049	1571.099976	1571.099976	5571915
2012	2023-02-22	1535.0	1560.000000	1381.199951	1404.849976	1404.849976	10606476
2013	2023-02-23	1380.0	1438.000000	1350.000000	1382.650024	1382.650024	8907540
2014	2023-02-24	1410.0	1427.000000	1261.599976	1315.650024	1315.650024	8736727
2015	2023-02-27	1300.0	1313.800049	1131.050049	1193.500000	1193.500000	10271008

EDA

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

There is no need of Adj Close

In [ ]:

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

Out[ ]:

	Date	Open	High	Low	Close	Volume
0	2015-01-01	74.399712	75.495628	73.663994	75.104774	3946806
1	2015-01-02	75.304039	76.177704	75.104774	75.472641	6565229
2	2015-01-05	75.273384	77.641479	75.212074	76.721832	9404837
3	2015-01-06	75.963120	79.381149	74.215782	76.139381	18412441
4	2015-01-07	76.637527	77.794754	73.878578	75.464973	10863352
...	...	...	...	...	...	...
2011	2023-02-21	1626.000000	1644.449951	1561.300049	1571.099976	5571915
2012	2023-02-22	1535.000000	1560.000000	1381.199951	1404.849976	10606476
2013	2023-02-23	1380.000000	1438.000000	1350.000000	1382.650024	8907540
2014	2023-02-24	1410.000000	1427.000000	1261.599976	1315.650024	8736727
2015	2023-02-27	1300.000000	1313.800049	1131.050049	1193.500000	10271008

2016 rows × 6 columns

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

Out[ ]:

Date	Open	High	Low	Close	Volume

In [ ]:
```
# Finding null values, if any
adani_1.isnull().sum()
```

Out[ ]:
```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

In [ ]:
```
#To check which rows have any missing value under any column
adani_1[adani_1.isnull().any(axis=1)]
```

Out[ ]:

Date	Open	High	Low	Close	Volume

In [ ]:
```
# Removing the row which have null value
adani_2=adani_1.dropna().reset_index(drop=True)
adani_2
```

Out[ ]:

	Date	Open	High	Low	Close	Volume
0	2015-01-01	74.399712	75.495628	73.663994	75.104774	3946806
1	2015-01-02	75.304039	76.177704	75.104774	75.472641	6565229
2	2015-01-05	75.273384	77.641479	75.212074	76.721832	9404837
3	2015-01-06	75.963120	79.381149	74.215782	76.139381	18412441
4	2015-01-07	76.637527	77.794754	73.878578	75.464973	10863352
...	...	...	...	...	...	...
2011	2023-02-21	1626.000000	1644.449951	1561.300049	1571.099976	5571915
2012	2023-02-22	1535.000000	1560.000000	1381.199951	1404.849976	10606476
2013	2023-02-23	1380.000000	1438.000000	1350.000000	1382.650024	8907540
2014	2023-02-24	1410.000000	1427.000000	1261.599976	1315.650024	8736727
2015	2023-02-27	1300.000000	1313.800049	1131.050049	1193.500000	10271008

2016 rows × 6 columns

In [ ]:

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

Out[ ]:

Date	Open	High	Low	Close	Volume

In [ ]:

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

Out[ ]:

	Date	Open	High	Low	Close	Volume
0	2015-01-01	74.399712	75.495628	73.663994	75.104774	3946806
1	2015-01-02	75.304039	76.177704	75.104774	75.472641	6565229
2	2015-01-05	75.273384	77.641479	75.212074	76.721832	9404837
3	2015-01-06	75.963120	79.381149	74.215782	76.139381	18412441
4	2015-01-07	76.637527	77.794754	73.878578	75.464973	10863352
...	...	...	...	...	...	...
2011	2023-02-21	1626.000000	1644.449951	1561.300049	1571.099976	5571915
2012	2023-02-22	1535.000000	1560.000000	1381.199951	1404.849976	10606476
2013	2023-02-23	1380.000000	1438.000000	1350.000000	1382.650024	8907540
2014	2023-02-24	1410.000000	1427.000000	1261.599976	1315.650024	8736727
2015	2023-02-27	1300.000000	1313.800049	1131.050049	1193.500000	10271008

2016 rows × 6 columns

Descriptive Statistics

In [ ]:
```
adani.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2016 entries, 0 to 2015
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    2016 non-null   datetime64[ns]
 1   Open    2016 non-null   float64
 2   High    2016 non-null   float64
 3   Low     2016 non-null   float64
 4   Close   2016 non-null   float64
 5   Volume  2016 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 94.6 KB
```

In [ ]:
```
adani.describe()
```

Out[ ]:

	Open	High	Low	Close	Volume
count	2016.000000	2016.000000	2016.000000	2016.000000	2.016000e+03
mean	623.985137	635.819773	610.334685	623.187029	8.029662e+06
std	982.032743	997.320182	962.574070	980.014466	1.056264e+07
min	32.149174	33.402447	31.713251	32.012947	2.482490e+05
25%	65.687843	67.302114	63.957783	65.510746	3.094653e+06
50%	137.228462	140.949997	133.375000	137.149994	5.206722e+06
75%	766.500000	802.237503	736.187500	780.625015	8.917879e+06
max	4175.000000	4190.000000	4066.399902	4165.299805	1.701502e+08

In [ ]:
```
adani.corr()
```

Out[ ]:

	Open	High	Low	Close	Volume
Open	1.000000	0.999775	0.999112	0.999145	-0.215898
High	0.999775	1.000000	0.998991	0.999400	-0.212922
Low	0.999112	0.998991	1.000000	0.999698	-0.221773
Close	0.999145	0.999400	0.999698	1.000000	-0.217414
Volume	-0.215898	-0.212922	-0.221773	-0.217414	1.000000

- Every attributes are highly correlated except volume

In [ ]:
```
# converting the date column in to datetime
adani['Date']=pd.to_datetime(adani['Date'],format='%Y-%m-%d')
adani
```

Out[ ]:

	Date	Open	High	Low	Close	Volume
0	2015-01-01	74.399712	75.495628	73.663994	75.104774	3946806
1	2015-01-02	75.304039	76.177704	75.104774	75.472641	6565229
2	2015-01-05	75.273384	77.641479	75.212074	76.721832	9404837
3	2015-01-06	75.963120	79.381149	74.215782	76.139381	18412441
4	2015-01-07	76.637527	77.794754	73.878578	75.464973	10863352
...	...	...	...	...	...	...
2011	2023-02-21	1626.000000	1644.449951	1561.300049	1571.099976	5571915
2012	2023-02-22	1535.000000	1560.000000	1381.199951	1404.849976	10606476
2013	2023-02-23	1380.000000	1438.000000	1350.000000	1382.650024	8907540
2014	2023-02-24	1410.000000	1427.000000	1261.599976	1315.650024	8736727
2015	2023-02-27	1300.000000	1313.800049	1131.050049	1193.500000	10271008

2016 rows × 6 columns

In [ ]:

```
# Setting the date column as index
adani=adani.set_index('Date')
adani
```

Out[ ]:

	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

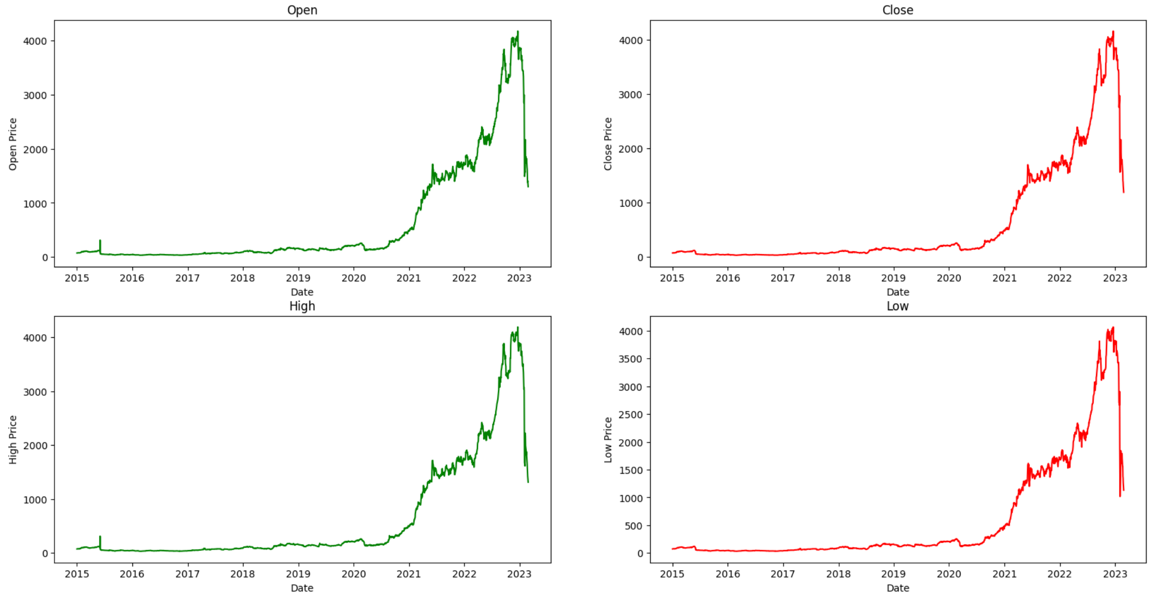
2016 rows × 5 columns

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 Stock

In [ ]:
```
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')
```

Out[ ]:
```
Text(0.5, 1.0, 'Low')
```



Box Plots are a popular data visualization tool used in stock analysis to depict the distribution of a dataset, particularly for numerical variables such as stock prices or returns.

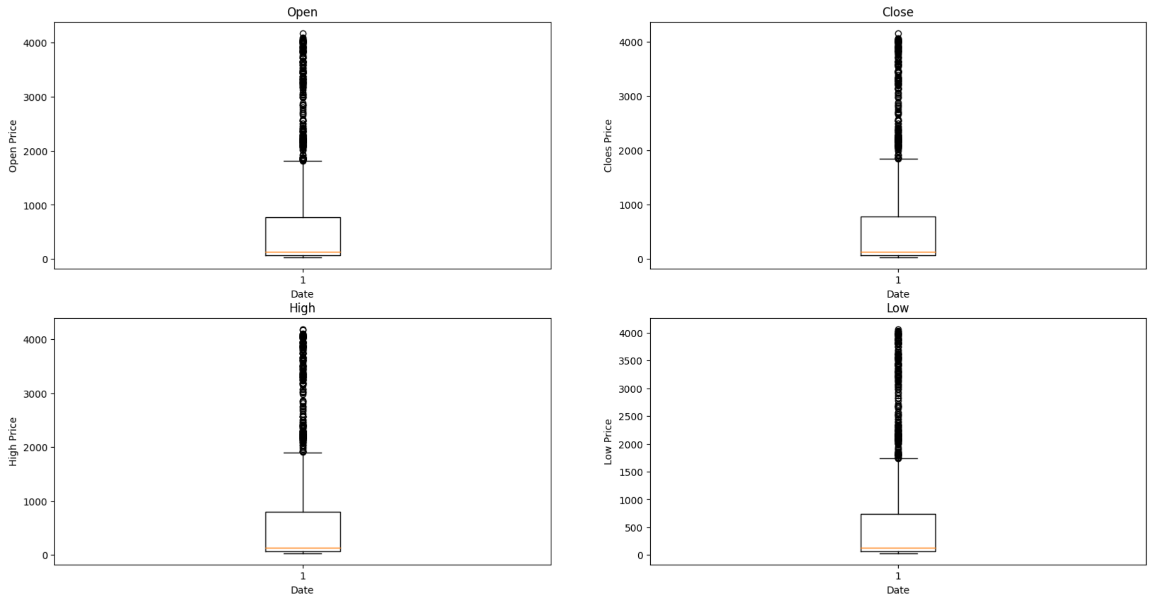
The following code-

- The subplot() function creates four subplots, arranged in a 2x2 grid.
- For each subplot, the *plt.boxplot()* function is used to plot the respective price data as a box plot.
- The whiskers of the box plot extend from the box to show the range of the data outside the IQR. Any data points outside the whiskers are considered outliers and

plotted separately as individual points.

In [ ]:
```
# 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')
```

Out[ ]:
```
Text(0.5, 1.0, 'Low')
```



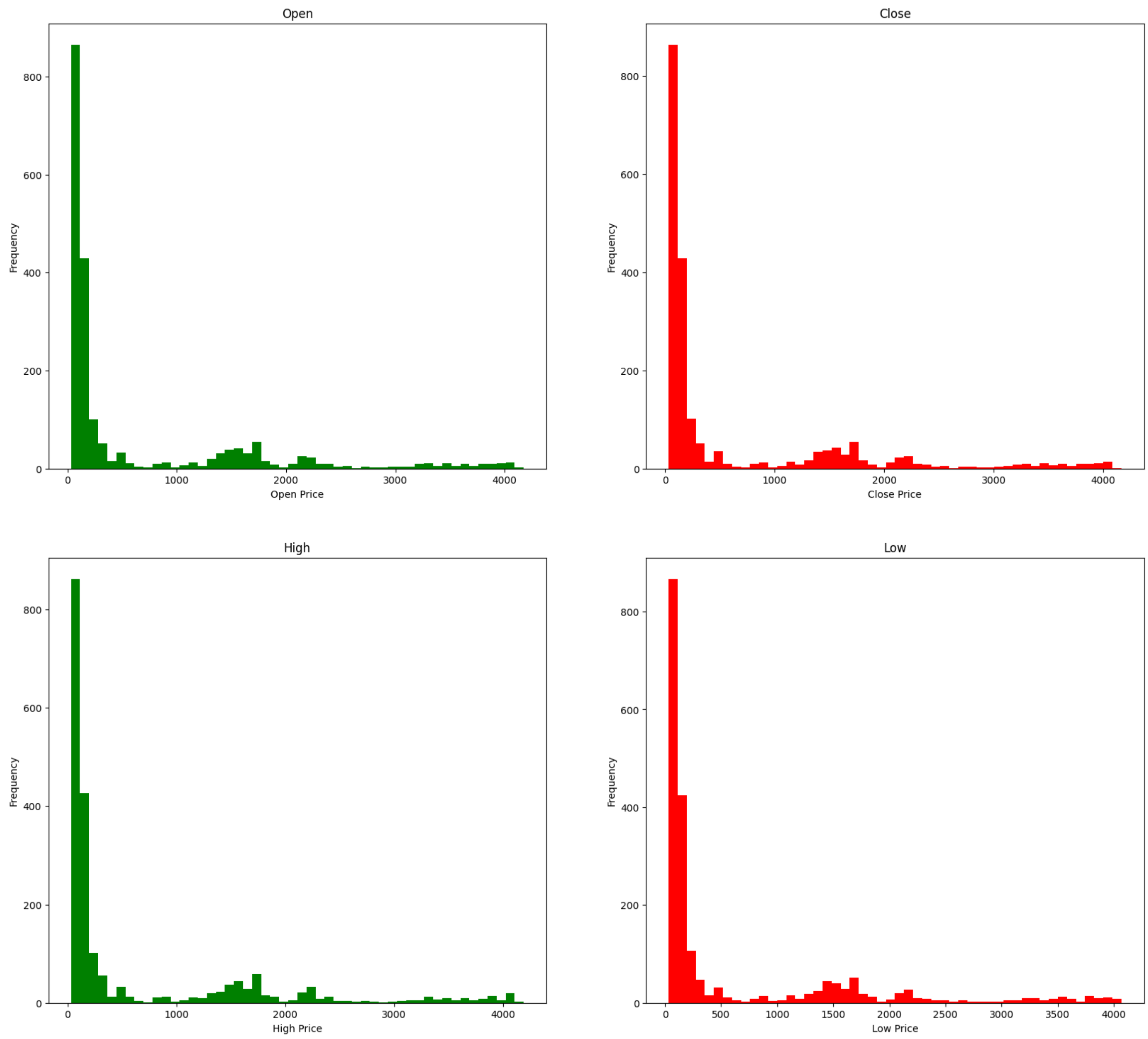
From the box plot it is clear that there are no outliers inthe dataset

- The following code creates four histograms of the daily stock prices of Adani Enterprises, each for the Open, Close, High, and Low prices, respectively.

In [ ]:
```
# 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')
```

```
Out[ ]: Text(0.5, 1.0, 'Low')
```

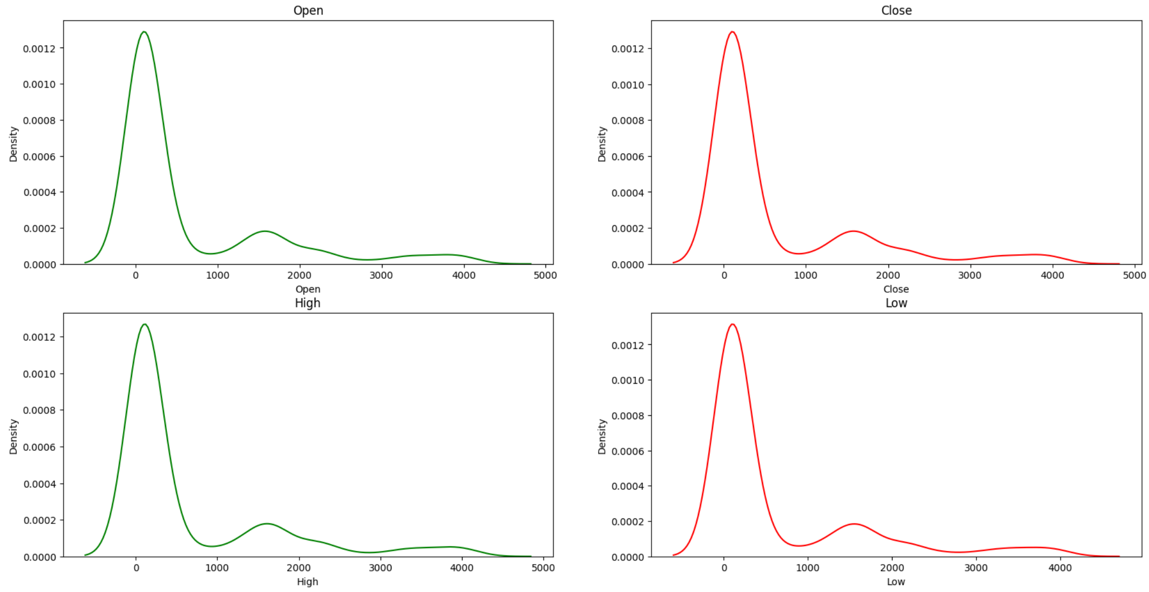


- A Kernel Density Estimation (KDE) plot is a type of data visualization that can be useful in stock analysis.

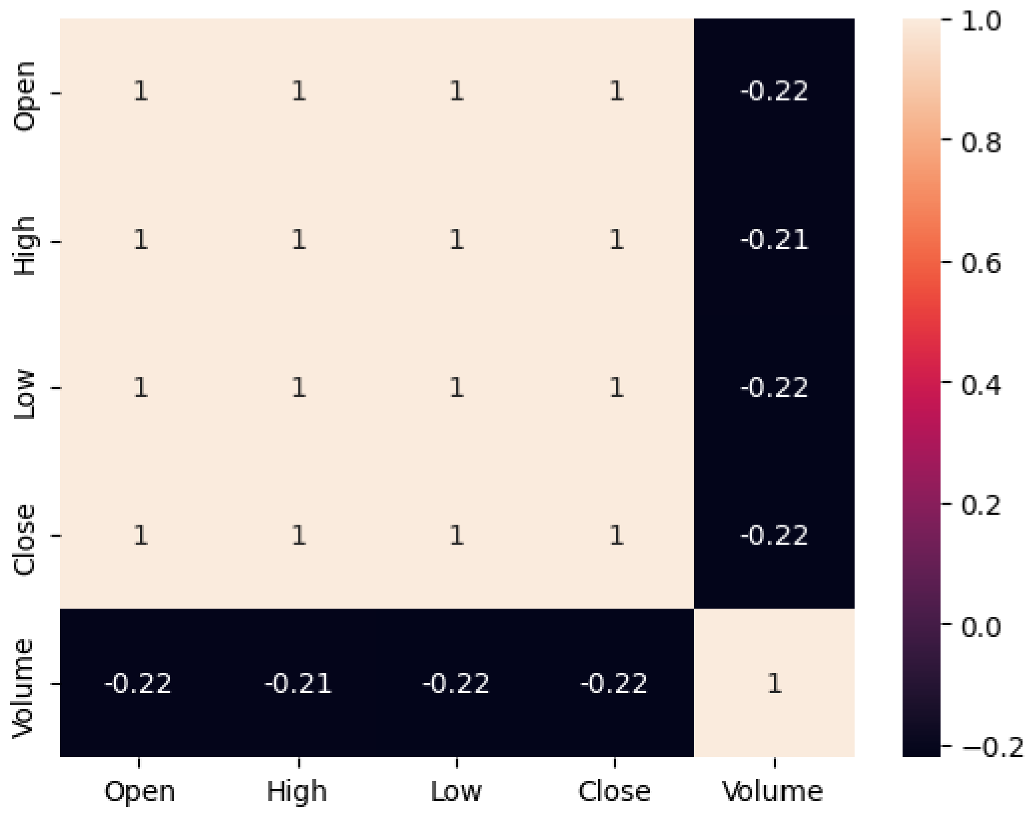
- KDE plots are essentially smoothed histograms that can provide insights into the underlying distribution of a dataset, such as stock prices or returns.

In [ ]:
```
# 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')
```

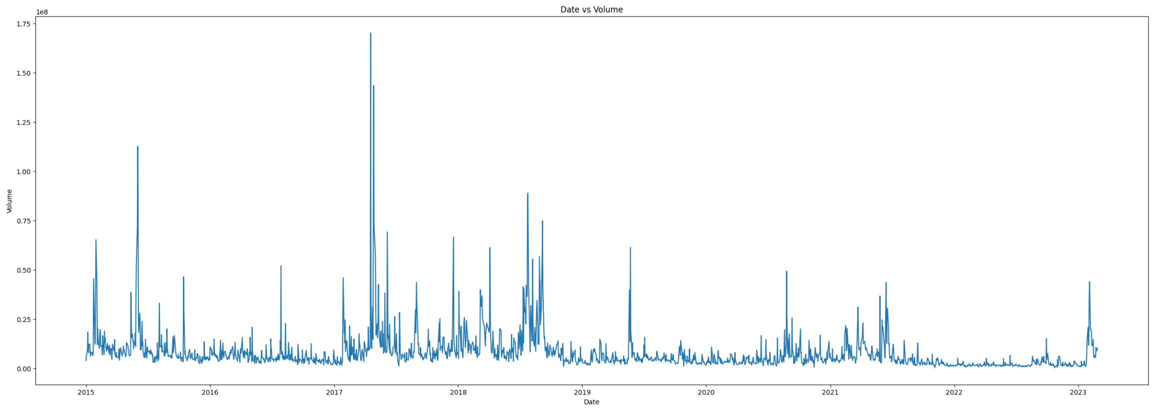
Out[ ]:
```
Text(0.5, 1.0, 'Low')
```



In [ ]:
```
sns.heatmap(adani.corr(),annot=True)
plt.show()
```



In [ ]:
```
figure=plt.figure(figsize=(30,10))
plt.plot(adani['Volume'])
plt.xlabel('Date')
plt.ylabel('Volume')
plt.title('Date vs Volume')
plt.show()
```



Finding long-term and short-term trends

Moving Average

In [ ]:
```
adani_ma=adani.copy()
adani_ma['30-day MA']=adani['Close'].rolling(window=30).mean()
adani_ma['200-day MA']=adani['Close'].rolling(window=200).mean()
```

In [ ]:
```
adani_ma
```

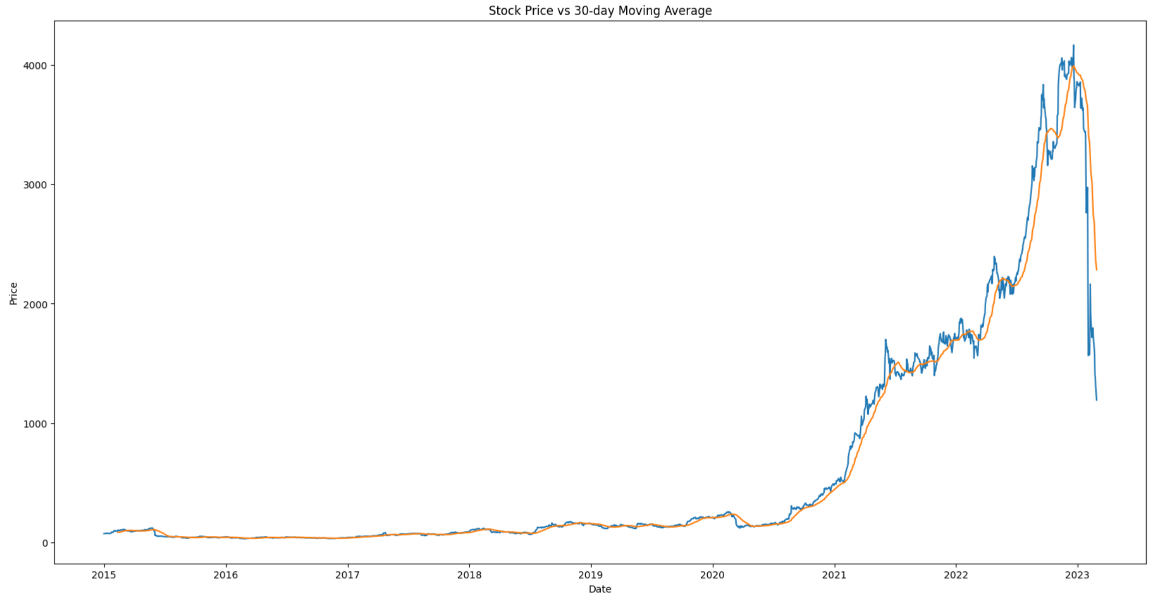
Out[ ]:

Date	Open	High	Low	Close	Volume	30-day MA	200-day MA
2015-01-01	74.399712	75.495628	73.663994	75.104774	3946806	NaN	NaN
2015-01-02	75.304039	76.177704	75.104774	75.472641	6565229	NaN	NaN
2015-01-05	75.273384	77.641479	75.212074	76.721832	9404837	NaN	NaN
2015-01-06	75.963120	79.381149	74.215782	76.139381	18412441	NaN	NaN
2015-01-07	76.637527	77.794754	73.878578	75.464973	10863352	NaN	NaN
...	...	...	...	...	...	...	...
2023-02-21	1626.000000	1644.449951	1561.300049	1571.099976	5571915	2596.011654	3015.067501
2023-02-22	1535.000000	1560.000000	1381.199951	1404.849976	10606476	2521.276656	3011.103501
2023-02-23	1380.000000	1438.000000	1350.000000	1382.650024	8907540	2446.171655	3007.469251
2023-02-24	1410.000000	1427.000000	1261.599976	1315.650024	8736727	2368.453324	3003.451001
2023-02-27	1300.000000	1313.800049	1131.050049	1193.500000	10271008	2284.198328	2999.192501

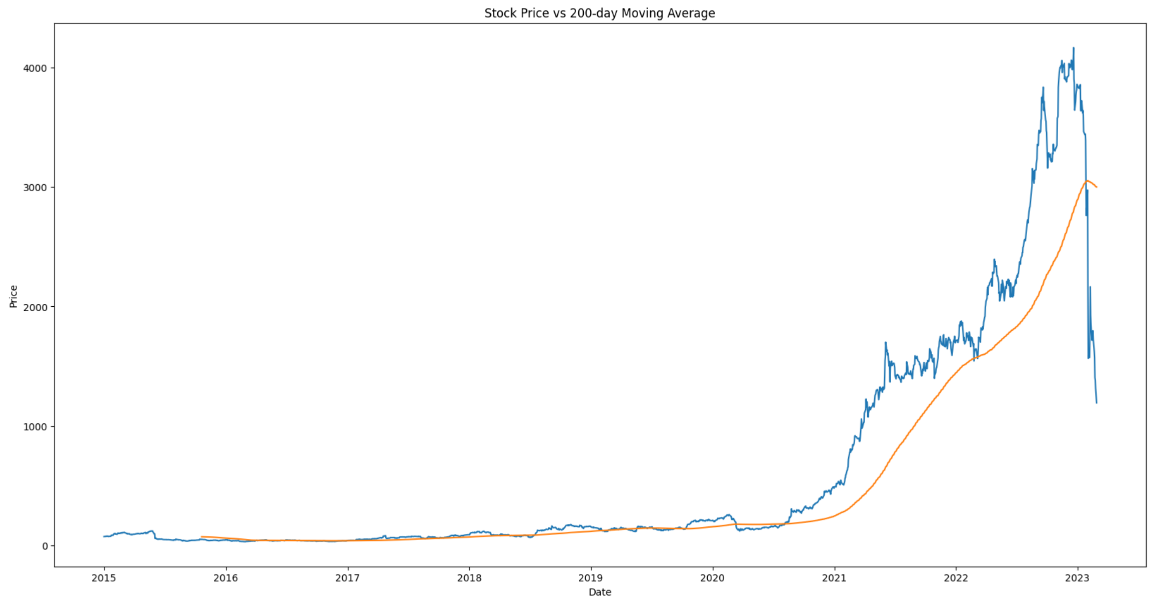
2016 rows × 7 columns

In [ ]:

```
plt.figure(figsize=(20,10))
plt.plot(adani_ma['Close'],label='Original data')
plt.plot(adani_ma['30-day MA'],label='30-MA')
plt.legend
plt.title('Stock Price vs 30-day Moving Average')
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()
```




```
In [ ]: plt.figure(figsize=(20,10))
        plt.plot(adani_ma['Close'],label='Original data')
        plt.plot(adani_ma['200-day MA'],label='200-MA')
        plt.legend
        plt.title('Stock Price vs 200-day Moving Average')
        plt.xlabel('Date')
        plt.ylabel('Price')
        plt.show()
```



Long term and short term trends can be identified using the Moving Average graphs

- In long term, Stock price is in upward trend
- Short term trends can be identified from MA-30 chart
- Stock had a major short term downtrend during the year 2020
- It may be due to the bearish market during the Covid-19 outbraek

Model Building

```
In [ ]: from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_v
        from sklearn.metrics import mean_poisson_deviance, mean_gamma_deviance, accuracy
```

```
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
```

In [ ]:

```
adani
```

Out[ ]:

	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

In [ ]:

```
# Creating dataframe which only includes date and close time

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

Out[ ]:

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

In [ ]:

```
print(close_df.shape)
```

```
(2016, 1)
```

In [ ]:

```
close_df=close_df.reset_index()
```

In [ ]:

```
close_df['Date']
```

Out[ ]:

```
0      2015-01-01
1      2015-01-02
2      2015-01-05
3      2015-01-06
4      2015-01-07
          ...
2011   2023-02-21
2012   2023-02-22
2013   2023-02-23
2014   2023-02-24
2015   2023-02-27
Name: Date, Length: 2016, dtype: datetime64[ns]
```

Normalizing / scaling close value between 0 to 1

In [ ]:

```
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

- Ratio for training and testing data is 86:14

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

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

Create new dataset according to requirement of time-series prediction

In [ ]:
```
# convert 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   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

In [ ]:
```
# reshape 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,)
```

Algorithms

Support vector regression - SVR

In [ ]:
```
import math
from sklearn.svm import SVR

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

Out[ ]:

▾ SVR

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

In [ ]:
```
# 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)
```

In [ ]:

```
# Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

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.

In [ ]:

```
# Evaluation metrices RMSE and MAE
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_pr
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_predi
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:  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.

In [ ]:

```
print("Train data explained variance regression score:", explained_variance_scor
print("Test data explained variance regression score:", explained_variance_score
```

```
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's explained by an independent variable or variables in a regression model.

1 = Best

0 or < 0 = worse

In [ ]:
```
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
```

Comparision between original stock close price vs predicted close price

In [ ]:
```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_pre
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test predic

plotdf = pd.DataFrame({'Date': close_stock['Date'],
                       'original_close': close_stock['Close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0]
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].t

fig = px.line(plotdf,x=plotdf['Date'], y=[plotdf['original_close'],plotdf['train
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','Date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predic
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

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

Predicting next 30 days

```
In [ ]: x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
        temp_input=list(x_input)
        temp_input=temp_input[0].tolist()

        from numpy import array

        lst_output=[]
        n_steps=time_step
        i=0
        pred_days = 30
        while(i<pred_days):

            if(len(temp_input)>time_step):

                x_input=np.array(temp_input[1:])
                #print("{} day input {}".format(i,x_input))
                x_input=x_input.reshape(1,-1)

                yhat = svr_rbf.predict(x_input)
                #print("{} day output {}".format(i,yhat))
                temp_input.extend(yhat.tolist())
                temp_input=temp_input[1:]

                lst_output.extend(yhat.tolist())
                i=i+1

            else:
                yhat = svr_rbf.predict(x_input)

                temp_input.extend(yhat.tolist())
                lst_output.extend(yhat.tolist())

                i=i+1

        print("Output of predicted next days: ", len(lst_output))
```

```
Output of predicted next days:  30
```

Plotting last 15 days and next predicted 30 days

```
In [ ]: 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]
```

```
In [ ]: temp_mat = np.empty((len(last_days)+pred_days+1,1))
        temp_mat[:] = np.nan
        temp_mat = temp_mat.reshape(1,-1).tolist()[0]

        last_original_days_value = temp_mat
        next_predicted_days_value = temp_mat

        last_original_days_value[0:time_step+1] = scaler.inverse_transform(closedf[len(c
        next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.array(lst_
```

```
new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})

names = cycle(['Last 15 days close price','Predicted next 30 days close price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index, y=[new_pred_plot['last_origir
                                                      new_pred_plot['next_predic
              labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 30 days',
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

Plotting whole closing stock price with prediction

In [ ]:
```
svrdf=closedf.tolist()
svrdf.extend((np.array(lst_output).reshape(-1,1)).tolist())
svrdf=scaler.inverse_transform(svrdf).reshape(1,-1).tolist()[0]

names = cycle(['Close Price'])

fig = px.line(svrdf,labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing stock price with predictior
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

Random Forest Regressor - RF

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
regressor.fit(X_train, y_train)
```

Out[ ]:
```
▾ RandomForestRegressor
RandomForestRegressor(random_state=0)
```

In [ ]:
```
# Lets Do the prediction

train_predict=regressor.predict(X_train)
test_predict=regressor.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)
```

In [ ]:
```
# Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

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.

In [ ]:
```
# Evaluation metrices RMSE and MAE
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_pr
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_predi
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:  6.566053542987049
Train data MSE:  43.113059129372786
Test data MAE:  2.468926710491606
-------------------------------------------------------------------------------
------
Test data RMSE:  1321.8514180025982
Test data MSE:  1747291.1712754795
Test data MAE:  1044.6012653745058
```

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.

In [ ]:
```
print("Train data explained variance regression score:", explained_variance_scor
print("Test data explained variance regression score:", explained_variance_score
```

```
Train data explained variance regression score: 0.9997854608474905
Test data explained variance regression score: 0.018458236415465845
```

R2 score for regression

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

1 = Best 0 or < 0 = worse

In [ ]:
```
train_r2_rf=r2_score(original_ytrain, train_predict)
test_r2_rf=r2_score(original_ytest, test_predict)
print("Train data R2 score:", train_r2_rf)
print("Test data R2 score:", test_r2_rf)
```

```
Train data R2 score: 0.9997854563518825
Test data R2 score: -1.5525401133123253
```

Comparision between original stock close price vs predicted close price

In [ ]:
```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_pre
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test predic


plotdf = pd.DataFrame({'Date': close_stock['Date'],
                       'original_close': close_stock['Close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0]
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].t

fig = px.line(plotdf,x=plotdf['Date'], y=[plotdf['original_close'],plotdf['trair
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','Date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predic
                  plot_bgcolor='white', font_size=15, font_color='black', legend
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

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

Predicting next 30 days

In [ ]:
```
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()

from numpy import array

lst_output=[]
n_steps=time_step
```

```
i=0
pred_days = 30
while(i<pred_days):

    if(len(temp_input)>time_step):

        x_input=np.array(temp_input[1:])
        #print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)

        yhat = regressor.predict(x_input)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat.tolist())
        temp_input=temp_input[1:]

        lst_output.extend(yhat.tolist())
        i=i+1

    else:
        yhat = regressor.predict(x_input)

        temp_input.extend(yhat.tolist())
        lst_output.extend(yhat.tolist())

        i=i+1

print("Output of predicted next days: ", len(lst_output))
```

```
Output of predicted next days:  30
```

Plotting last 15 days and next predicted 30 days

In [ ]:
```
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]
```

In [ ]:
```
temp_mat = np.empty((len(last_days)+pred_days+1,1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1,-1).tolist()[0]

last_original_days_value = temp_mat
next_predicted_days_value = temp_mat

last_original_days_value[0:time_step+1] = scaler.inverse_transform(closedf[len(c
next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.array(lst_

names = cycle(['Last 15 days close price','Predicted next 30 days close price'])

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})

fig = px.line(new_pred_plot,x=new_pred_plot.index, y=[new_pred_plot['last_origin
                                                      new_pred_plot['next_predic
```

```
                labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 30 days',
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

Plotting whole closing stock price with prediction

In [ ]:
```
rfdf=closedf.tolist()
rfdf.extend((np.array(lst_output).reshape(-1,1)).tolist())
rfdf=scaler.inverse_transform(rfdf).reshape(1,-1).tolist()[0]

names = cycle(['Close price'])

fig = px.line(rfdf,labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing stock price with predictior
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

K-nearest neighbour - KNN

In [ ]:
```
from sklearn import neighbors

K = time_step
neighbor = neighbors.KNeighborsRegressor(n_neighbors = K)
neighbor.fit(X_train, y_train)
```

Out[ ]:
```
▾        KNeighborsRegressor
KNeighborsRegressor(n_neighbors=13)
```

In [ ]:
```
# Lets Do the prediction

train_predict=neighbor.predict(X_train)
test_predict=neighbor.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)
```

In [ ]:
```
# Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
```

```
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

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.

In [ ]:
```
# Evaluation metrices RMSE and MAE
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_pr
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_predi
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:  19.046264942714977
Train data MSE:  362.7602082680936
Test data MAE:  8.124832189509325
-------------------------------------------------------------------------------
------
Test data RMSE:  1328.3165462241775
Test data MSE:  1764424.8469729272
Test data MAE:  1053.7853376624716
```

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.

In [ ]:
```
print("Train data explained variance regression score:", explained_variance_scor
print("Test data explained variance regression score:", explained_variance_score
```

```
Train data explained variance regression score: 0.9982047772817602
Test data explained variance regression score: -0.004799251384312031
```

R2 score for regression

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

1 = Best 0 or < 0 = worse

In [ ]:
```
train_r2_knn=r2_score(original_ytrain, train_predict)
test_r2_knn=r2_score(original_ytest, test_predict)
print("Train data R2 score:", train_r2_knn)
print("Test data R2 score:", test_r2_knn)
```

```
Train data R2 score: 0.998194795265162
Test data R2 score: -1.5775699396087033
```

Comparision between original stock close price vs predicted close price

In [ ]:

```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_pre
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test predic

plotdf = pd.DataFrame({'Date': close_stock['Date'],
                       'original_close': close_stock['Close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0]
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].t

fig = px.line(plotdf,x=plotdf['Date'], y=[plotdf['original_close'],plotdf['trair
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','Date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predic
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

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

Predicting next 30 days

In [ ]:

```
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()

from numpy import array

lst_output=[]
n_steps=time_step
i=0
pred_days = 30
while(i<pred_days):

    if(len(temp_input)>time_step):

        x_input=np.array(temp_input[1:])
```

```
        #print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)

        yhat = neighbor.predict(x_input)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat.tolist())
        temp_input=temp_input[1:]

        lst_output.extend(yhat.tolist())
        i=i+1

    else:
        yhat = neighbor.predict(x_input)

        temp_input.extend(yhat.tolist())
        lst_output.extend(yhat.tolist())

        i=i+1

print("Output of predicted next days: ", len(lst_output))
```

```
Output of predicted next days:  30
```

Plotting last 15 days and next predicted 30 days

In [ ]:

```
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]
```

In [ ]:

```
temp_mat = np.empty((len(last_days)+pred_days+1,1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1,-1).tolist()[0]

last_original_days_value = temp_mat
next_predicted_days_value = temp_mat

last_original_days_value[0:time_step+1] = scaler.inverse_transform(closedf[len(c
next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.array(lst_

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})

names = cycle(['Last 15 days close price','Predicted next 30 days close price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index, y=[new_pred_plot['last_origin
                                                      new_pred_plot['next_predic
              labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 30 days',
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
```

```
fig.update_yaxes(showgrid=False)
fig.show()
```

Plotting whole closing stock price with prediction

In [ ]:
```
knndf=closedf.tolist()
knndf.extend((np.array(lst_output).reshape(-1,1)).tolist())
knndf=scaler.inverse_transform(knndf).reshape(1,-1).tolist()[0]

names = cycle(['Close price'])

fig = px.line(knndf,labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing stock price with predictior
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

LSTM

In [ ]:
```
# 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)
```

LSTM model structure

In [ ]:
```
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')
```

In [ ]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 13, 32)            4352

 lstm_1 (LSTM)               (None, 13, 32)            8320

 lstm_2 (LSTM)               (None, 32)                8320

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 21,025
Trainable params: 21,025
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 13, 32)            4352

 lstm_1 (LSTM)               (None, 13, 32)            8320

 lstm_2 (LSTM)               (None, 32)                8320

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 21,025
Trainable params: 21,025
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

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

```
Epoch 1/100
54/54 [==============================] - 10s 36ms/step - loss: 0.0033 - val_los
s: 0.0094
Epoch 2/100
54/54 [==============================] - 1s 10ms/step - loss: 9.5820e-05 - val_
loss: 0.0078
Epoch 3/100
54/54 [==============================] - 0s 9ms/step - loss: 8.5011e-05 - val_l
oss: 0.0083
Epoch 4/100
54/54 [==============================] - 1s 9ms/step - loss: 8.9847e-05 - val_l
oss: 0.0112
Epoch 5/100
54/54 [==============================] - 1s 13ms/step - loss: 9.0348e-05 - val_
loss: 0.0150
Epoch 6/100
54/54 [==============================] - 1s 11ms/step - loss: 8.0884e-05 - val_
loss: 0.0143
Epoch 7/100
54/54 [==============================] - 1s 10ms/step - loss: 7.8615e-05 - val_
loss: 0.0189
Epoch 8/100
54/54 [==============================] - 1s 11ms/step - loss: 8.2998e-05 - val_
loss: 0.0191
Epoch 9/100
54/54 [==============================] - 1s 10ms/step - loss: 8.5144e-05 - val_
loss: 0.0175
Epoch 10/100
54/54 [==============================] - 1s 11ms/step - loss: 7.8632e-05 - val_
loss: 0.0168
Epoch 11/100
54/54 [==============================] - 1s 11ms/step - loss: 8.9257e-05 - val_
loss: 0.0178
Epoch 12/100
54/54 [==============================] - 1s 12ms/step - loss: 9.6739e-05 - val_
loss: 0.0211
Epoch 13/100
54/54 [==============================] - 1s 12ms/step - loss: 7.6754e-05 - val_
loss: 0.0137
Epoch 14/100
54/54 [==============================] - 1s 12ms/step - loss: 8.0549e-05 - val_
loss: 0.0147
Epoch 15/100
54/54 [==============================] - 1s 10ms/step - loss: 8.1249e-05 - val_
loss: 0.0206
Epoch 16/100
54/54 [==============================] - 0s 9ms/step - loss: 8.6908e-05 - val_l
oss: 0.0193
Epoch 17/100
54/54 [==============================] - 1s 9ms/step - loss: 8.9014e-05 - val_l
oss: 0.0151
Epoch 18/100
54/54 [==============================] - 1s 10ms/step - loss: 7.6697e-05 - val_
loss: 0.0183
Epoch 19/100
54/54 [==============================] - 1s 10ms/step - loss: 7.6858e-05 - val_
loss: 0.0159
Epoch 20/100
54/54 [==============================] - 1s 10ms/step - loss: 8.0679e-05 - val_
loss: 0.0159
```

```
Epoch 21/100
54/54 [==============================] - 1s 9ms/step - loss: 6.7815e-05 - val_l
oss: 0.0146
Epoch 22/100
54/54 [==============================] - 1s 12ms/step - loss: 8.8865e-05 - val_
loss: 0.0201
Epoch 23/100
54/54 [==============================] - 1s 11ms/step - loss: 7.0295e-05 - val_
loss: 0.0158
Epoch 24/100
54/54 [==============================] - 1s 11ms/step - loss: 7.2771e-05 - val_
loss: 0.0148
Epoch 25/100
54/54 [==============================] - 1s 12ms/step - loss: 8.0682e-05 - val_
loss: 0.0288
Epoch 26/100
54/54 [==============================] - 1s 10ms/step - loss: 8.2901e-05 - val_
loss: 0.0163
Epoch 27/100
54/54 [==============================] - 1s 12ms/step - loss: 7.7988e-05 - val_
loss: 0.0197
Epoch 28/100
54/54 [==============================] - 1s 12ms/step - loss: 7.4064e-05 - val_
loss: 0.0181
Epoch 29/100
54/54 [==============================] - 1s 12ms/step - loss: 6.7871e-05 - val_
loss: 0.0139
Epoch 30/100
54/54 [==============================] - 1s 12ms/step - loss: 6.9761e-05 - val_
loss: 0.0150
Epoch 31/100
54/54 [==============================] - 1s 11ms/step - loss: 7.9246e-05 - val_
loss: 0.0156
Epoch 32/100
54/54 [==============================] - 1s 11ms/step - loss: 6.7068e-05 - val_
loss: 0.0142
Epoch 33/100
54/54 [==============================] - 1s 12ms/step - loss: 7.1763e-05 - val_
loss: 0.0226
Epoch 34/100
54/54 [==============================] - 1s 12ms/step - loss: 7.7572e-05 - val_
loss: 0.0113
Epoch 35/100
54/54 [==============================] - 1s 10ms/step - loss: 6.9528e-05 - val_
loss: 0.0190
Epoch 36/100
54/54 [==============================] - 1s 9ms/step - loss: 6.7232e-05 - val_l
oss: 0.0138
Epoch 37/100
54/54 [==============================] - 1s 9ms/step - loss: 7.9096e-05 - val_l
oss: 0.0147
Epoch 38/100
54/54 [==============================] - 1s 9ms/step - loss: 8.8393e-05 - val_l
oss: 0.0183
Epoch 39/100
54/54 [==============================] - 1s 10ms/step - loss: 6.4870e-05 - val_
loss: 0.0211
Epoch 40/100
54/54 [==============================] - 1s 10ms/step - loss: 6.4623e-05 - val_
loss: 0.0153
```

```
Epoch 41/100
54/54 [==============================] - 1s 10ms/step - loss: 6.2081e-05 - val_
loss: 0.0203
Epoch 42/100
54/54 [==============================] - 1s 10ms/step - loss: 7.1308e-05 - val_
loss: 0.0151
Epoch 43/100
54/54 [==============================] - 1s 10ms/step - loss: 6.7679e-05 - val_
loss: 0.0142
Epoch 44/100
54/54 [==============================] - 1s 12ms/step - loss: 6.9296e-05 - val_
loss: 0.0176
Epoch 45/100
54/54 [==============================] - 1s 10ms/step - loss: 5.9843e-05 - val_
loss: 0.0153
Epoch 46/100
54/54 [==============================] - 1s 12ms/step - loss: 6.1121e-05 - val_
loss: 0.0122
Epoch 47/100
54/54 [==============================] - 1s 11ms/step - loss: 5.7803e-05 - val_
loss: 0.0135
Epoch 48/100
54/54 [==============================] - 1s 11ms/step - loss: 5.9771e-05 - val_
loss: 0.0121
Epoch 49/100
54/54 [==============================] - 1s 9ms/step - loss: 5.8081e-05 - val_l
oss: 0.0154
Epoch 50/100
54/54 [==============================] - 1s 10ms/step - loss: 6.3766e-05 - val_
loss: 0.0131
Epoch 51/100
54/54 [==============================] - 1s 10ms/step - loss: 6.3711e-05 - val_
loss: 0.0132
Epoch 52/100
54/54 [==============================] - 1s 10ms/step - loss: 5.5235e-05 - val_
loss: 0.0123
Epoch 53/100
54/54 [==============================] - 0s 9ms/step - loss: 5.9406e-05 - val_l
oss: 0.0108
Epoch 54/100
54/54 [==============================] - 1s 11ms/step - loss: 5.1200e-05 - val_
loss: 0.0147
Epoch 55/100
54/54 [==============================] - 1s 10ms/step - loss: 5.6665e-05 - val_
loss: 0.0122
Epoch 56/100
54/54 [==============================] - 1s 10ms/step - loss: 6.6170e-05 - val_
loss: 0.0129
Epoch 57/100
54/54 [==============================] - 1s 11ms/step - loss: 4.8908e-05 - val_
loss: 0.0131
Epoch 58/100
54/54 [==============================] - 1s 10ms/step - loss: 5.5806e-05 - val_
loss: 0.0080
Epoch 59/100
54/54 [==============================] - 1s 10ms/step - loss: 4.7699e-05 - val_
loss: 0.0156
Epoch 60/100
54/54 [==============================] - 1s 10ms/step - loss: 5.2780e-05 - val_
loss: 0.0105
```

```
Epoch 61/100
54/54 [==============================] - 1s 10ms/step - loss: 8.0774e-05 - val_
loss: 0.0110
Epoch 62/100
54/54 [==============================] - 1s 10ms/step - loss: 5.5566e-05 - val_
loss: 0.0084
Epoch 63/100
54/54 [==============================] - 1s 11ms/step - loss: 5.4960e-05 - val_
loss: 0.0116
Epoch 64/100
54/54 [==============================] - 1s 12ms/step - loss: 4.3047e-05 - val_
loss: 0.0104
Epoch 65/100
54/54 [==============================] - 1s 11ms/step - loss: 4.6060e-05 - val_
loss: 0.0112
Epoch 66/100
54/54 [==============================] - 1s 11ms/step - loss: 4.3948e-05 - val_
loss: 0.0106
Epoch 67/100
54/54 [==============================] - 1s 12ms/step - loss: 4.9347e-05 - val_
loss: 0.0105
Epoch 68/100
54/54 [==============================] - 1s 14ms/step - loss: 4.8002e-05 - val_
loss: 0.0096
Epoch 69/100
54/54 [==============================] - 1s 12ms/step - loss: 5.6641e-05 - val_
loss: 0.0109
Epoch 70/100
54/54 [==============================] - 1s 13ms/step - loss: 4.2549e-05 - val_
loss: 0.0113
Epoch 71/100
54/54 [==============================] - 1s 13ms/step - loss: 5.1489e-05 - val_
loss: 0.0057
Epoch 72/100
54/54 [==============================] - 1s 12ms/step - loss: 4.3126e-05 - val_
loss: 0.0076
Epoch 73/100
54/54 [==============================] - 1s 13ms/step - loss: 3.3828e-05 - val_
loss: 0.0106
Epoch 74/100
54/54 [==============================] - 1s 13ms/step - loss: 3.4438e-05 - val_
loss: 0.0082
Epoch 75/100
54/54 [==============================] - 1s 15ms/step - loss: 3.2152e-05 - val_
loss: 0.0057
Epoch 76/100
54/54 [==============================] - 1s 12ms/step - loss: 3.0747e-05 - val_
loss: 0.0062
Epoch 77/100
54/54 [==============================] - 1s 12ms/step - loss: 3.3167e-05 - val_
loss: 0.0047
Epoch 78/100
54/54 [==============================] - 1s 17ms/step - loss: 3.7350e-05 - val_
loss: 0.0064
Epoch 79/100
54/54 [==============================] - 1s 14ms/step - loss: 2.8867e-05 - val_
loss: 0.0086
Epoch 80/100
54/54 [==============================] - 1s 12ms/step - loss: 2.6284e-05 - val_
loss: 0.0048
```

```
Epoch 81/100
54/54 [==============================] - 1s 10ms/step - loss: 3.2426e-05 - val_
loss: 0.0040
Epoch 82/100
54/54 [==============================] - 1s 10ms/step - loss: 2.9465e-05 - val_
loss: 0.0054
Epoch 83/100
54/54 [==============================] - 1s 12ms/step - loss: 3.3215e-05 - val_
loss: 0.0075
Epoch 84/100
54/54 [==============================] - 1s 10ms/step - loss: 3.0282e-05 - val_
loss: 0.0063
Epoch 85/100
54/54 [==============================] - 1s 12ms/step - loss: 3.0921e-05 - val_
loss: 0.0030
Epoch 86/100
54/54 [==============================] - 1s 11ms/step - loss: 2.7025e-05 - val_
loss: 0.0060
Epoch 87/100
54/54 [==============================] - 1s 11ms/step - loss: 2.9181e-05 - val_
loss: 0.0061
Epoch 88/100
54/54 [==============================] - 1s 10ms/step - loss: 3.0448e-05 - val_
loss: 0.0049
Epoch 89/100
54/54 [==============================] - 1s 12ms/step - loss: 2.6746e-05 - val_
loss: 0.0052
Epoch 90/100
54/54 [==============================] - 1s 11ms/step - loss: 2.9661e-05 - val_
loss: 0.0060
Epoch 91/100
54/54 [==============================] - 1s 11ms/step - loss: 2.6171e-05 - val_
loss: 0.0074
Epoch 92/100
54/54 [==============================] - 1s 10ms/step - loss: 2.3744e-05 - val_
loss: 0.0053
Epoch 93/100
54/54 [==============================] - 1s 11ms/step - loss: 2.5040e-05 - val_
loss: 0.0035
Epoch 94/100
54/54 [==============================] - 1s 11ms/step - loss: 2.4155e-05 - val_
loss: 0.0066
Epoch 95/100
54/54 [==============================] - 1s 9ms/step - loss: 2.4909e-05 - val_l
oss: 0.0051
Epoch 96/100
54/54 [==============================] - 1s 11ms/step - loss: 2.3251e-05 - val_
loss: 0.0062
Epoch 97/100
54/54 [==============================] - 0s 9ms/step - loss: 2.4138e-05 - val_l
oss: 0.0067
Epoch 98/100
54/54 [==============================] - 1s 10ms/step - loss: 2.8247e-05 - val_
loss: 0.0071
Epoch 99/100
54/54 [==============================] - 0s 9ms/step - loss: 2.6753e-05 - val_l
oss: 0.0054
Epoch 100/100
54/54 [==============================] - 0s 9ms/step - loss: 2.3200e-05 - val_l
oss: 0.0064
```

Out[ ]: <keras.callbacks.History at 0x1cca30dd810>

In [ ]:
```
### 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
```

```
54/54 [==============================] - 2s 4ms/step
9/9 [==============================] - 0s 3ms/step
```

Out[ ]: ((1719, 1), (269, 1))

In [ ]:
```
# Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

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.

In [ ]:
```
# Evaluation metrices RMSE and MAE
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_pr
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_predi
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:  19.48636569773769
Train data MSE:  379.7184481059681
Test data MAE:  13.169414032053433
-------------------------------------------------------------------------------
------
Test data RMSE:  330.91655864505253
Test data MSE:  109505.7687854845
Test data MAE:  258.16923710138826
```

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.

In [ ]:
```
print("Train data explained variance regression score:", explained_variance_scor
print("Test data explained variance regression score:", explained_variance_score
```

```
Train data explained variance regression score: 0.9983609924872215
Test data explained variance regression score: 0.9171651612576752
```

R2 score for regression

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

1 = Best 0 or < 0 = worse

In [ ]:
```
train_r2_lstm=r2_score(original_ytrain, train_predict)
test_r2_lstm=r2_score(original_ytest, test_predict)
print("Train data R2 score:", train_r2_lstm)
print("Test data R2 score:", test_r2_lstm)
```

```
Train data R2 score: 0.9981104059243464
Test data R2 score: 0.8400278831260775
```

Comparision between original stock close price vs predicted close price

In [ ]:
```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_pre
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test predic


plotdf = pd.DataFrame({'Date': close_stock['Date'],
                       'original_close': close_stock['Close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0]
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].t

fig = px.line(plotdf,x=plotdf['Date'], y=[plotdf['original_close'],plotdf['trair
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','Date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predic
                  plot_bgcolor='white', font_size=15, font_color='black', legenc
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

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

Predicting next 30 days

```python
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()

from numpy import array

lst_output=[]
n_steps=time_step
i=0
pred_days = 30
while(i<pred_days):

    if(len(temp_input)>time_step):

        x_input=np.array(temp_input[1:])
        #print("{} day input {}".format(i,x_input))
        x_input = x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))

        yhat = model.predict(x_input, verbose=0)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)

        lst_output.extend(yhat.tolist())
        i=i+1

    else:

        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())

        lst_output.extend(yhat.tolist())
        i=i+1

print("Output of predicted next days: ", len(lst_output))
```

```
Output of predicted next days:  30
```

Plotting last 15 days and next predicted 30 days

```python
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]
```

```python
temp_mat = np.empty((len(last_days)+pred_days+1,1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1,-1).tolist()[0]

last_original_days_value = temp_mat
next_predicted_days_value = temp_mat
```

```
last_original_days_value[0:time_step+1] = scaler.inverse_transform(closedf[len(c
next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.array(lst_

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})

names = cycle(['Last 15 days close price','Predicted next 30 days close price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index, y=[new_pred_plot['last_origir
                                                      new_pred_plot['next_predic
              labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 30 days',
                  plot_bgcolor='white', font_size=15, font_color='black',legend_

fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

Plotting whole closing stock price with prediction

In [ ]:

```
lstmdf=closedf.tolist()
lstmdf.extend((np.array(lst_output).reshape(-1,1)).tolist())
lstmdf=scaler.inverse_transform(lstmdf).reshape(1,-1).tolist()[0]

names = cycle(['Close price'])

fig = px.line(lstmdf,labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing stock price with predictior
                  plot_bgcolor='white', font_size=15, font_color='black',legend_

fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

GRU (Gated Recurrent Unit)

In [ ]:

```
# 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)
```

In [ ]:

```
tf.keras.backend.clear_session()
model=Sequential()
model.add(GRU(32,return_sequences=True,input_shape=(time_step,1)))
model.add(GRU(32,return_sequences=True))
```

```
model.add(GRU(32,return_sequences=True))
model.add(GRU(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [ ]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 13, 32)            3360

 gru_1 (GRU)                 (None, 13, 32)            6336

 gru_2 (GRU)                 (None, 13, 32)            6336

 gru_3 (GRU)                 (None, 32)                6336

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 22,401
Trainable params: 22,401
Non-trainable params: 0
_________________________________________________________________
```


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 13, 32)            3360

 gru_1 (GRU)                 (None, 13, 32)            6336

 gru_2 (GRU)                 (None, 13, 32)            6336

 gru_3 (GRU)                 (None, 32)                6336

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 22,401
Trainable params: 22,401
Non-trainable params: 0
_________________________________________________________________
```


In [ ]:
```
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100,batch_size=
```

```
Epoch 1/100
54/54 [==============================] - 12s 46ms/step - loss: 0.0013 - val_los
s: 0.0045
Epoch 2/100
54/54 [==============================] - 1s 15ms/step - loss: 6.0047e-05 - val_
loss: 0.0037
Epoch 3/100
54/54 [==============================] - 1s 15ms/step - loss: 6.4925e-05 - val_
loss: 0.0047
Epoch 4/100
54/54 [==============================] - 1s 14ms/step - loss: 6.1997e-05 - val_
loss: 0.0034
Epoch 5/100
54/54 [==============================] - 1s 14ms/step - loss: 5.7345e-05 - val_
loss: 0.0041
Epoch 6/100
54/54 [==============================] - 1s 14ms/step - loss: 4.7953e-05 - val_
loss: 0.0032
Epoch 7/100
54/54 [==============================] - 1s 14ms/step - loss: 6.2636e-05 - val_
loss: 0.0062
Epoch 8/100
54/54 [==============================] - 1s 15ms/step - loss: 5.5959e-05 - val_
loss: 0.0033
Epoch 9/100
54/54 [==============================] - 1s 14ms/step - loss: 6.3823e-05 - val_
loss: 0.0034
Epoch 10/100
54/54 [==============================] - 1s 14ms/step - loss: 5.4235e-05 - val_
loss: 0.0039
Epoch 11/100
54/54 [==============================] - 1s 15ms/step - loss: 5.0347e-05 - val_
loss: 0.0034
Epoch 12/100
54/54 [==============================] - 1s 15ms/step - loss: 5.3633e-05 - val_
loss: 0.0032
Epoch 13/100
54/54 [==============================] - 1s 14ms/step - loss: 6.0883e-05 - val_
loss: 0.0072
Epoch 14/100
54/54 [==============================] - 1s 15ms/step - loss: 5.9821e-05 - val_
loss: 0.0057
Epoch 15/100
54/54 [==============================] - 1s 17ms/step - loss: 4.3307e-05 - val_
loss: 0.0036
Epoch 16/100
54/54 [==============================] - 1s 17ms/step - loss: 3.8040e-05 - val_
loss: 0.0046
Epoch 17/100
54/54 [==============================] - 1s 16ms/step - loss: 4.2344e-05 - val_
loss: 0.0025
Epoch 18/100
54/54 [==============================] - 1s 17ms/step - loss: 4.3954e-05 - val_
loss: 0.0050
Epoch 19/100
54/54 [==============================] - 1s 14ms/step - loss: 4.2869e-05 - val_
loss: 0.0035
Epoch 20/100
54/54 [==============================] - 1s 14ms/step - loss: 4.8379e-05 - val_
loss: 0.0019
```

```
Epoch 21/100
54/54 [==============================] - 1s 15ms/step - loss: 4.2996e-05 - val_
loss: 0.0031
Epoch 22/100
54/54 [==============================] - 1s 14ms/step - loss: 4.4934e-05 - val_
loss: 0.0037
Epoch 23/100
54/54 [==============================] - 1s 14ms/step - loss: 3.4045e-05 - val_
loss: 0.0028
Epoch 24/100
54/54 [==============================] - 1s 14ms/step - loss: 3.4310e-05 - val_
loss: 0.0045
Epoch 25/100
54/54 [==============================] - 1s 14ms/step - loss: 4.0114e-05 - val_
loss: 0.0047
Epoch 26/100
54/54 [==============================] - 1s 14ms/step - loss: 3.1658e-05 - val_
loss: 0.0035
Epoch 27/100
54/54 [==============================] - 1s 14ms/step - loss: 3.0156e-05 - val_
loss: 0.0030
Epoch 28/100
54/54 [==============================] - 1s 14ms/step - loss: 2.5620e-05 - val_
loss: 0.0032
Epoch 29/100
54/54 [==============================] - 1s 14ms/step - loss: 3.0462e-05 - val_
loss: 0.0045
Epoch 30/100
54/54 [==============================] - 1s 14ms/step - loss: 2.8245e-05 - val_
loss: 0.0045
Epoch 31/100
54/54 [==============================] - 1s 14ms/step - loss: 3.0139e-05 - val_
loss: 0.0023
Epoch 32/100
54/54 [==============================] - 1s 14ms/step - loss: 2.1358e-05 - val_
loss: 0.0038
Epoch 33/100
54/54 [==============================] - 1s 14ms/step - loss: 3.5671e-05 - val_
loss: 0.0045
Epoch 34/100
54/54 [==============================] - 1s 15ms/step - loss: 3.3183e-05 - val_
loss: 0.0036
Epoch 35/100
54/54 [==============================] - 1s 14ms/step - loss: 2.6575e-05 - val_
loss: 0.0045
Epoch 36/100
54/54 [==============================] - 1s 14ms/step - loss: 3.3264e-05 - val_
loss: 0.0046
Epoch 37/100
54/54 [==============================] - 1s 14ms/step - loss: 3.3997e-05 - val_
loss: 0.0033
Epoch 38/100
54/54 [==============================] - 1s 14ms/step - loss: 2.3737e-05 - val_
loss: 0.0033
Epoch 39/100
54/54 [==============================] - 1s 14ms/step - loss: 2.1650e-05 - val_
loss: 0.0021
Epoch 40/100
54/54 [==============================] - 1s 15ms/step - loss: 2.6256e-05 - val_
loss: 0.0035
```

```
Epoch 41/100
54/54 [==============================] - 1s 15ms/step - loss: 2.1778e-05 - val_
loss: 0.0037
Epoch 42/100
54/54 [==============================] - 1s 15ms/step - loss: 2.1561e-05 - val_
loss: 0.0021
Epoch 43/100
54/54 [==============================] - 1s 14ms/step - loss: 2.7380e-05 - val_
loss: 0.0030
Epoch 44/100
54/54 [==============================] - 1s 14ms/step - loss: 1.9332e-05 - val_
loss: 0.0021
Epoch 45/100
54/54 [==============================] - 1s 14ms/step - loss: 2.2651e-05 - val_
loss: 0.0030
Epoch 46/100
54/54 [==============================] - 1s 14ms/step - loss: 2.2340e-05 - val_
loss: 0.0026
Epoch 47/100
54/54 [==============================] - 1s 15ms/step - loss: 2.2685e-05 - val_
loss: 0.0019
Epoch 48/100
54/54 [==============================] - 1s 15ms/step - loss: 2.2159e-05 - val_
loss: 0.0028
Epoch 49/100
54/54 [==============================] - 1s 14ms/step - loss: 1.9052e-05 - val_
loss: 0.0029
Epoch 50/100
54/54 [==============================] - 1s 14ms/step - loss: 2.1211e-05 - val_
loss: 0.0023
Epoch 51/100
54/54 [==============================] - 1s 14ms/step - loss: 3.2906e-05 - val_
loss: 0.0026
Epoch 52/100
54/54 [==============================] - 1s 14ms/step - loss: 2.3377e-05 - val_
loss: 0.0022
Epoch 53/100
54/54 [==============================] - 1s 14ms/step - loss: 2.1617e-05 - val_
loss: 0.0021
Epoch 54/100
54/54 [==============================] - 1s 14ms/step - loss: 1.8003e-05 - val_
loss: 0.0039
Epoch 55/100
54/54 [==============================] - 1s 14ms/step - loss: 3.0621e-05 - val_
loss: 0.0025
Epoch 56/100
54/54 [==============================] - 1s 14ms/step - loss: 3.2953e-05 - val_
loss: 0.0023
Epoch 57/100
54/54 [==============================] - 1s 14ms/step - loss: 2.7016e-05 - val_
loss: 0.0016
Epoch 58/100
54/54 [==============================] - 1s 14ms/step - loss: 3.3563e-05 - val_
loss: 0.0033
Epoch 59/100
54/54 [==============================] - 1s 15ms/step - loss: 2.1391e-05 - val_
loss: 0.0017
Epoch 60/100
54/54 [==============================] - 1s 15ms/step - loss: 1.9404e-05 - val_
loss: 0.0023
```

```
Epoch 61/100
54/54 [==============================] - 1s 16ms/step - loss: 1.8988e-05 - val_
loss: 0.0023
Epoch 62/100
54/54 [==============================] - 1s 16ms/step - loss: 1.8037e-05 - val_
loss: 0.0026
Epoch 63/100
54/54 [==============================] - 1s 16ms/step - loss: 2.7479e-05 - val_
loss: 0.0018
Epoch 64/100
54/54 [==============================] - 1s 14ms/step - loss: 1.8213e-05 - val_
loss: 0.0045
Epoch 65/100
54/54 [==============================] - 1s 15ms/step - loss: 2.9048e-05 - val_
loss: 0.0024
Epoch 66/100
54/54 [==============================] - 1s 14ms/step - loss: 2.5011e-05 - val_
loss: 0.0038
Epoch 67/100
54/54 [==============================] - 1s 13ms/step - loss: 2.0914e-05 - val_
loss: 0.0016
Epoch 68/100
54/54 [==============================] - 1s 17ms/step - loss: 2.4347e-05 - val_
loss: 0.0014
Epoch 69/100
54/54 [==============================] - 1s 13ms/step - loss: 2.7244e-05 - val_
loss: 0.0022
Epoch 70/100
54/54 [==============================] - 1s 12ms/step - loss: 3.1558e-05 - val_
loss: 0.0040
Epoch 71/100
54/54 [==============================] - 1s 13ms/step - loss: 1.9915e-05 - val_
loss: 0.0017
Epoch 72/100
54/54 [==============================] - 1s 15ms/step - loss: 2.0076e-05 - val_
loss: 0.0025
Epoch 73/100
54/54 [==============================] - 1s 13ms/step - loss: 2.2293e-05 - val_
loss: 0.0025
Epoch 74/100
54/54 [==============================] - 1s 12ms/step - loss: 1.8247e-05 - val_
loss: 0.0030
Epoch 75/100
54/54 [==============================] - 1s 13ms/step - loss: 2.0855e-05 - val_
loss: 0.0027
Epoch 76/100
54/54 [==============================] - 1s 13ms/step - loss: 1.8738e-05 - val_
loss: 0.0031
Epoch 77/100
54/54 [==============================] - 1s 13ms/step - loss: 2.3482e-05 - val_
loss: 0.0024
Epoch 78/100
54/54 [==============================] - 1s 14ms/step - loss: 2.0249e-05 - val_
loss: 0.0029
Epoch 79/100
54/54 [==============================] - 1s 12ms/step - loss: 1.8111e-05 - val_
loss: 0.0024
Epoch 80/100
54/54 [==============================] - 1s 12ms/step - loss: 2.1178e-05 - val_
loss: 0.0030
```

```
Epoch 81/100
54/54 [==============================] - 1s 15ms/step - loss: 1.8050e-05 - val_
loss: 0.0031
Epoch 82/100
54/54 [==============================] - 1s 17ms/step - loss: 1.9902e-05 - val_
loss: 0.0038
Epoch 83/100
54/54 [==============================] - 1s 15ms/step - loss: 2.3675e-05 - val_
loss: 0.0027
Epoch 84/100
54/54 [==============================] - 1s 15ms/step - loss: 2.0077e-05 - val_
loss: 0.0020
Epoch 85/100
54/54 [==============================] - 1s 14ms/step - loss: 2.6431e-05 - val_
loss: 0.0015
Epoch 86/100
54/54 [==============================] - 1s 12ms/step - loss: 2.5610e-05 - val_
loss: 0.0016
Epoch 87/100
54/54 [==============================] - 1s 14ms/step - loss: 2.1006e-05 - val_
loss: 0.0020
Epoch 88/100
54/54 [==============================] - 1s 15ms/step - loss: 1.8027e-05 - val_
loss: 0.0042
Epoch 89/100
54/54 [==============================] - 1s 15ms/step - loss: 2.2194e-05 - val_
loss: 0.0027
Epoch 90/100
54/54 [==============================] - 1s 15ms/step - loss: 2.0436e-05 - val_
loss: 0.0042
Epoch 91/100
54/54 [==============================] - 1s 17ms/step - loss: 1.7975e-05 - val_
loss: 0.0035
Epoch 92/100
54/54 [==============================] - 1s 18ms/step - loss: 2.8020e-05 - val_
loss: 0.0021
Epoch 93/100
54/54 [==============================] - 1s 16ms/step - loss: 2.4421e-05 - val_
loss: 0.0023
Epoch 94/100
54/54 [==============================] - 1s 20ms/step - loss: 2.4989e-05 - val_
loss: 0.0025
Epoch 95/100
54/54 [==============================] - 1s 20ms/step - loss: 2.3643e-05 - val_
loss: 0.0028
Epoch 96/100
54/54 [==============================] - 1s 21ms/step - loss: 1.9171e-05 - val_
loss: 0.0034
Epoch 97/100
54/54 [==============================] - 1s 15ms/step - loss: 2.0692e-05 - val_
loss: 0.0022
Epoch 98/100
54/54 [==============================] - 1s 18ms/step - loss: 1.9615e-05 - val_
loss: 0.0033
Epoch 99/100
54/54 [==============================] - 1s 16ms/step - loss: 2.0781e-05 - val_
loss: 0.0028
Epoch 100/100
54/54 [==============================] - 1s 18ms/step - loss: 2.4081e-05 - val_
loss: 0.0027
```

```
Out[ ]: <keras.callbacks.History at 0x1ccaed55d10>
```

```
In [ ]: ### 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
```

```
54/54 [==============================] - 2s 4ms/step
9/9 [==============================] - 0s 5ms/step
Out[ ]: ((1719, 1), (269, 1))
```

```
In [ ]: # Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

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.

```
In [ ]: # Evaluation metrices RMSE and MAE
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_pr
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_predi
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:  19.164770894311555
Train data MSE:  367.28844343145136
Test data MAE:  14.672072061624133
------------------------------------------------------------------------------
------
Test data RMSE:  215.81286455296404
Test data MSE:  46575.192506556006
Test data MAE:  160.51109387706202
```

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.

```
In [ ]: print("Train data explained variance regression score:", explained_variance_scor
print("Test data explained variance regression score:", explained_variance_score
```

```
Train data explained variance regression score: 0.9989022447496304
Test data explained variance regression score: 0.9566045945222258
```

R2 score for regression

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

1 = Best 0 or < 0 = worse

In [ ]:
```
train_r2_gru=r2_score(original_ytrain, train_predict)
test_r2_gru=r2_score(original_ytest, test_predict)
print("Train data R2 score:", train_r2_gru)
print("Test data R2 score:", test_r2_gru)
```

```
Train data R2 score: 0.9981722613946573
Test data R2 score: 0.9319603686479772
```

Comparision between original stock close price vs predicted close price

In [ ]:
```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_pre
print("Test predicted data: ", testPredictPlot.shape)


names = cycle(['Original close price','Train predicted close price','Test predic

plotdf = pd.DataFrame({'Date': close_stock['Date'],
                       'original_close': close_stock['Close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0]
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].t

fig = px.line(plotdf,x=plotdf['Date'], y=[plotdf['original_close'],plotdf['trair
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','Date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predic
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

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

In [ ]:
```
plotdf.head(100)
```

Out[ ]:

	Date	original_close	train_predicted_close	test_predicted_close
0	2015-01-01	75.104774	NaN	NaN
1	2015-01-02	75.472641	NaN	NaN
2	2015-01-05	76.721832	NaN	NaN
3	2015-01-06	76.139381	NaN	NaN
4	2015-01-07	75.464973	NaN	NaN
...	...	...	...	...
95	2015-05-25	120.520180	133.397095	NaN
96	2015-05-26	121.623756	132.916519	NaN
97	2015-05-27	121.163933	133.746109	NaN
98	2015-05-28	112.580528	133.615875	NaN
99	2015-05-29	108.258171	127.419449	NaN

100 rows × 4 columns

Predicting next 30 days

In [ ]:

```
x_input=test_data[len(test_data)-time_step:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()

from numpy import array

lst_output=[]
n_steps=time_step
i=0
pred_days = 30
while(i<pred_days):

    if(len(temp_input)>time_step):

        x_input=np.array(temp_input[1:])
        #print("{} day input {}".format(i,x_input))
        x_input = x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))

        yhat = model.predict(x_input, verbose=0)
        #print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)

        lst_output.extend(yhat.tolist())
        i=i+1

    else:

        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())
```

```
        lst_output.extend(yhat.tolist())
        i=i+1

print("Output of predicted next days: ", len(lst_output))
```

```
Output of predicted next days:  30
```

Plotting last 15 days and next predicted 30 days

In [ ]:
```
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]
```

In [ ]:
```
temp_mat = np.empty((len(last_days)+pred_days+1,1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1,-1).tolist()[0]

last_original_days_value = temp_mat
next_predicted_days_value = temp_mat

last_original_days_value[0:time_step+1] = scaler.inverse_transform(closedf[len(c
next_predicted_days_value[time_step+1:] = scaler.inverse_transform(np.array(lst_

new_pred_plot = pd.DataFrame({
    'last_original_days_value':last_original_days_value,
    'next_predicted_days_value':next_predicted_days_value
})
names = cycle(['Last 15 days close price','Predicted next 30 days close price'])

fig = px.line(new_pred_plot,x=new_pred_plot.index, y=[new_pred_plot['last_origin
                                                      new_pred_plot['next_predic
              labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next 30 days',
                  plot_bgcolor='white', font_size=15, font_color='black', legend
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

Plotting whole closing stock price with prediction

In [ ]:
```
grudf=closedf.tolist()
grudf.extend((np.array(lst_output).reshape(-1,1)).tolist())
grudf=scaler.inverse_transform(grudf).reshape(1,-1).tolist()[0]

names = cycle(['Close price'])
fig = px.line(grudf,labels={'value': 'Stock price','index': 'Timestamp'})
fig.update_layout(title_text='Plotting whole closing stock price with predictior
                  plot_bgcolor='white', font_size=15, font_color='black',legend_
fig.for_each_trace(lambda t:  t.update(name = next(names)))
```

```
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

In [ ]:
```
finaldf = pd.DataFrame({
    'svr':svrdf,
    'rf':rfdf,
    'knn':knndf,
    'lstm':lstmdf,
    'gru':grudf,
})
finaldf
```

Out[ ]:

	svr	rf	knn	lstm	gru
0	75.104774	75.104774	75.104774	75.104774	75.104774
1	75.472641	75.472641	75.472641	75.472641	75.472641
2	76.721832	76.721832	76.721832	76.721832	76.721832
3	76.139381	76.139381	76.139381	76.139381	76.139381
4	75.464973	75.464973	75.464973	75.464973	75.464973
...	...	...	...	...	...
2041	1201.867428	1513.258010	1563.788452	1483.057085	1668.732820
2042	1198.824278	1511.278005	1567.046124	1482.188286	1679.166666
2043	1195.459067	1503.810009	1568.049974	1480.388603	1689.201898
2044	1192.325639	1500.590505	1565.123056	1478.261628	1698.844056
2045	1189.410871	1497.178000	1559.353825	1476.300208	1708.099302

2046 rows × 5 columns

Conclusion Chart

In [ ]:
```
names = cycle(['SVR', 'RF','KNN','LSTM','GRU'])

fig = px.line(finaldf[225:], x=finaldf.index[225:], y=[finaldf['svr'][225:],fina
                                                finaldf['lstm'][225:], finaldf['gru'][
             labels={'x': 'Timestamp','value':'Stock close price'})
fig.update_layout(title_text='Final stock analysis chart', font_size=15, font_co
fig.for_each_trace(lambda t:  t.update(name = next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)

fig.show()
```

In [ ]:
```
data={"Model": ["SVR", "Random Forest", "KNN", "LSTM", "GRU"],
    "Train R2 Score": [train_r2_svr, train_r2_rf,train_r2_knn, train_r2_lstm, tr
    "Test R2 Score": [test_r2_svr, test_r2_rf, test_r2_knn, test_r2_lstm, test_r
df=pd.DataFrame(data)
df
```

Out[ ]:

	Model	Train R2 Score	Test R2 Score
0	SVR	0.434754	0.224490
1	Random Forest	0.999785	-1.552540
2	KNN	0.998195	-1.577570
3	LSTM	0.998110	0.840028
4	GRU	0.998172	0.931960

By Looking into this table we can say that our LSTM model have best R2 score.

so we are going to use LSTM model for our deployment part.

In [ ]:

```
```