Project Name:- Predict Stock Market on Tesla Dataset By Random Forest and Decison tree

```
In statistics, Random Forest and Decison tree is an approach for modelling
the relationship between a scalar dependt variable y and one more
explanatory variables (or independent variables)denoted X.The case of onr
explanatory vaiables is called simple Random Forest and Decison tree
```

Here is the formal defintion "Linear regression is an approach for modelling the reletionship between a scalar dependent variables y and one or more explanatory variables (or independent variables)denoted x"

Let me explain the concept of regression in a very basic manner, so imagine that you run a company that builds cars and you want to understand how the change in prices of raw materials (let's say Steel) will affect the sales of the car. The general understanding is this, the rise in the price of steel will lead to a rise in the price of the car resulting in lesser demand and in turn lesser ales. But how do we quantify this? And how do we predict how much change in sales will happen based on the degree of change in steel price. That's when the regression comes

Random Forest and Decison tree is the analysis of two separate variables to define a single relationship and is a useful measure for technical and quantitative analysis in financial markets.

Plotting stock prices along a normal distribution-bell curve-can allow traders to see when a stock is overbought or oversold.

I Using Random Forest and Decison tree, a trader can identify key price points-entry price, stop-loss price, and exit prices.

A stock's price and time period determine the system parameters for linear regression, making the method universally applicable. Stock market close price is an important piece of information that is very useful for every short-term trader. The close prices are very important, especially for swing traders and position traders.

In this case study we choose Random Forest and Decison tree for our analysis. First, we divide the data into two parts of training and testing. Then we use the training section for starting analysis and defining the model.

Opening price

The opening price is the value that each share has the opening price gives a good indication of where the stock will move during the day. Since the Stock exchange can be likened with an auction market i.e. buyers and sellers meet to make deals with the highest bidder, the

opening price does not have to be the same as the last day's closing price.

Closing Price

An adjusted closing price is a stock's closing price on any given day of trading that has been amended to include any distributions and corporate actions that occurred at any time prior to the next day's open. The adjusted closing price is often used when examining historical returns or performing a detailed analysis on historical returns

Volume

Volume is one of the most basic and beneficial concepts to understand when trading stocks. Volume is defined as, "the number of shares or contracts traded in a security or an entire market during a given period of time"

ML02 Project:- Predict Stock Mraket

Objective : Best Models for Close price prediction

Md Sharfe Alam (90749)

Project - Purpose : Prediction of Close price of Tesla

Stocks

Steps and Tasks :

Step 1 : Reading and understanding of data

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import linear_model
%matplotlib inline
from sklearn import metrics
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [2]:
```
# library to ignore warnings that arise during visualizations
import warnings
warnings.filterwarnings('ignore')
```

Data Cleaning and Preparation

In [3]:
```
data=pd.read_csv("C:\\Users\\mdsha\\Downloads\\archive (3)\TSLA.csv")#reading
```

In [4]:
```
data.head(10)#checking out data
```

Out[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	3.800	5.000	3.508	4.778	4.778	93831500
1	2010-06-30	5.158	6.084	4.660	4.766	4.766	85935500
2	2010-07-01	5.000	5.184	4.054	4.392	4.392	41094000
3	2010-07-02	4.600	4.620	3.742	3.840	3.840	25699000
4	2010-07-06	4.000	4.000	3.166	3.222	3.222	34334500
5	2010-07-07	3.280	3.326	2.996	3.160	3.160	34608500
6	2010-07-08	3.228	3.504	3.114	3.492	3.492	38557000
7	2010-07-09	3.516	3.580	3.310	3.480	3.480	20253000
8	2010-07-12	3.590	3.614	3.400	3.410	3.410	11012500
9	2010-07-13	3.478	3.728	3.380	3.628	3.628	13400500

In [5]:
```
data.shape #checking shape of the dataset
```

Out[5]: (2956, 7)

In [6]:
```
data.drop('Adj Close',axis=1,inplace=True) #Drop the unnecessary column
```

In [7]:
```
data.head() #again it print 5 rows by default
```

Out[7]:

	Date	Open	High	Low	Close	Volume
0	2010-06-29	3.800	5.000	3.508	4.778	93831500
1	2010-06-30	5.158	6.084	4.660	4.766	85935500
2	2010-07-01	5.000	5.184	4.054	4.392	41094000
3	2010-07-02	4.600	4.620	3.742	3.840	25699000
4	2010-07-06	4.000	4.000	3.166	3.222	34334500

In [8]:
```
data.isnull().sum() #to check datatypes and null values
```

Out[8]:
```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

In [9]:
```
data.isna().any()#Cheking how many values are missing in the dataset
```

Out[9]:
```
Date      False
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

In [10]:
```
data.info() # getting basic data info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2956 entries, 0 to 2955
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    2956 non-null   object 
 1   Open    2956 non-null   float64
 2   High    2956 non-null   float64
 3   Low     2956 non-null   float64
 4   Close   2956 non-null   float64
 5   Volume  2956 non-null   int64  
dtypes: float64(4), int64(1), object(1)
memory usage: 138.7+ KB
```

In [11]:
```
data.describe() # to check for outliers
```

Out[11]:

	Open	High	Low	Close	Volume
count	2956.000000	2956.000000	2956.000000	2956.000000	2.956000e+03
mean	138.691296	141.771603	135.425953	138.762183	3.131449e+07
std	250.044839	255.863239	243.774157	250.123115	2.798383e+07
min	3.228000	3.326000	2.996000	3.160000	5.925000e+05
25%	19.627000	20.402000	19.127500	19.615000	1.310288e+07
50%	46.656999	47.487001	45.820002	46.545000	2.488680e+07
75%	68.057001	69.357500	66.911501	68.103998	3.973875e+07
max	1234.410034	1243.489990	1217.000000	1229.910034	3.046940e+08

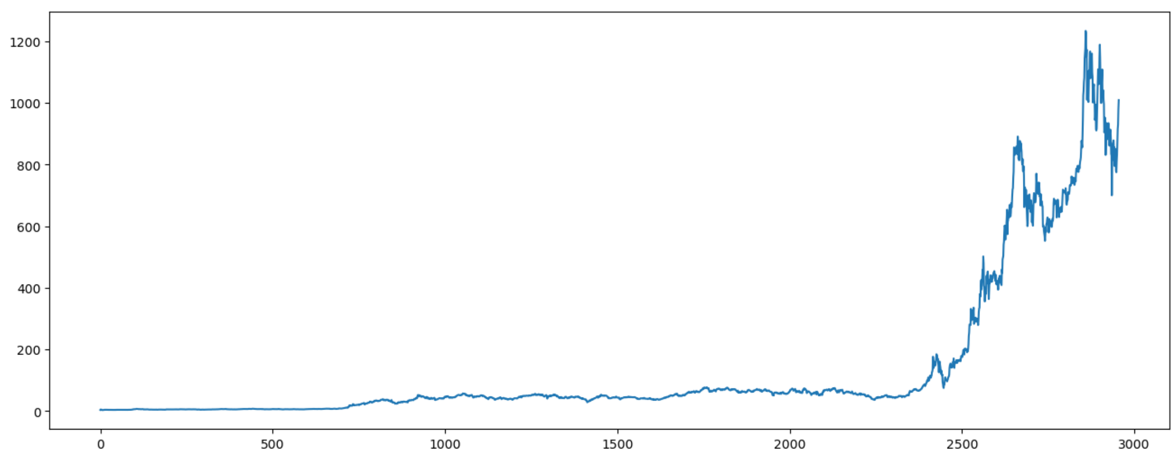
In [12]:
```
print(len(data)) #Print length of dataset
```

```
2956
```

Exploratly Data Analysis

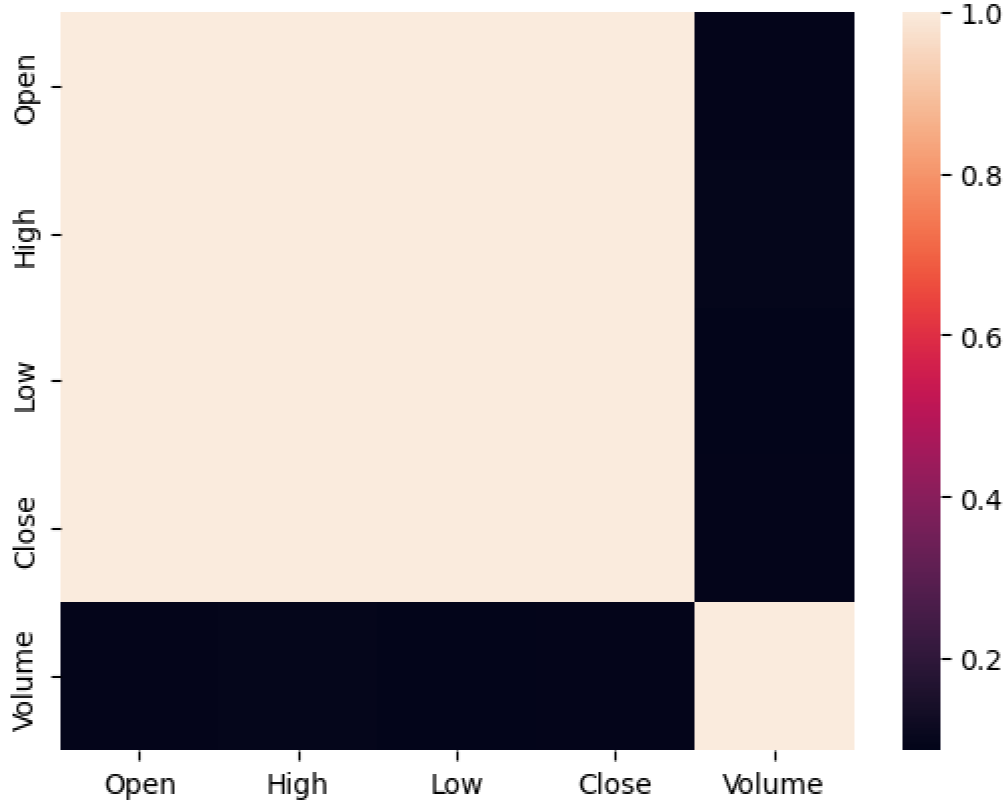
In [13]:
```
data['Open'].plot(figsize=(16,6))
```

Out[13]: <Axes: >



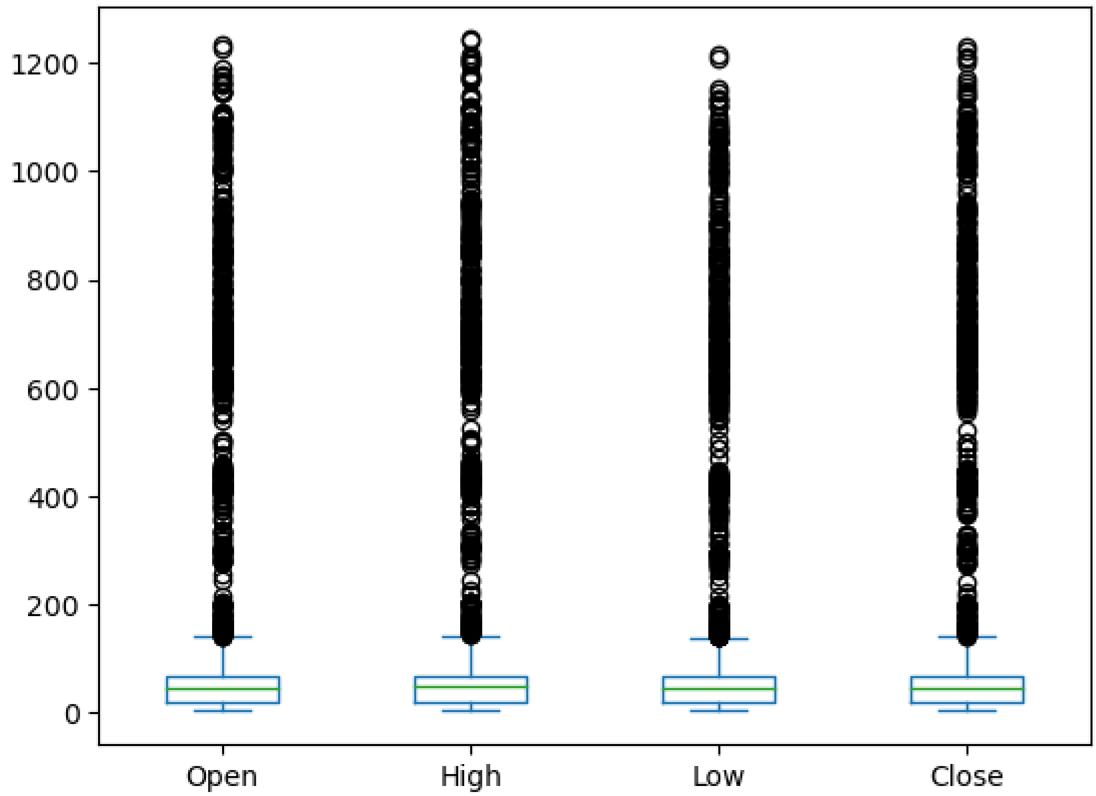
In [14]:
```
#This is a method that calculates the pairwise correlation between numerical c
sns.heatmap(data.corr())
```

Out[14]: <Axes: >



In [17]:
```
data[['Open','High','Low','Close']].plot(kind='box') #box plot
```

Out[17]: <Axes: >



Train-Test split and feature scalling

In [18]:
```
x=data[['Open','High','Low','Volume']]
y=data['Close']
```

In [19]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y, random_state=0)
```

In [20]:
```
x_train.shape
```

Out[20]: (2217, 4)

In [21]:
```
x_test.shape
```

Out[21]: (739, 4)

Model Building

Random Forest for Stock Price Prediction

In [22]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [23]:
```
rf_regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
rf_regressor.fit(x_train, y_train)
```

Out[23]:
```
RandomForestRegressor(random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluation And Accuracy

In [24]:
```
print("Train Accuracy :", rf_regressor.score(x_train, y_train))
print("Test Accuracy :", rf_regressor.score(x_test, y_test))
```

```
Train Accuracy : 0.999915304914194
Test Accuracy : 0.9995325615384516
```

In [25]:
```
predicted=rf_regressor.predict(x_test)
```

In [26]:
```
print(x_test)
```

```
            Open        High         Low    Volume
1749   74.884003   75.374001   70.959999  86307000
643     6.832000    6.970000    6.784000   7183500
118     5.734000    5.994000    5.706000   3714500
252     5.558000    5.650000    5.534000   4446000
1311   50.220001   50.849998   49.933998  14454500
...          ...         ...         ...       ...
794    31.400000   32.459999   31.000000  64659500
2429  167.800003  172.699997  164.440002  75961000
517     7.000000    7.042000    6.476000  12846500
1943   63.299999   64.150002   61.933998  37421500
1912   70.199997   71.931999   69.725998  20988500

[739 rows x 4 columns]
```

In [27]:
```
Df=pd.DataFrame(y_test,predicted)
```

In [28]:
```
Df=pd.DataFrame({'Actual price':y_test,'Predicted Price':predicted})
```

In [29]:
```
print(Df)
```

```
      Actual price  Predicted Price
1749     71.463997        73.091481
643       6.876000         6.916000
118       5.920000         5.883500
252       5.622000         5.612200
1311     50.638000        50.414980
...            ...              ...
794      32.368000        31.765420
2429    166.757996       167.381917
517       6.670000         6.815400
1943     62.712002        62.814939
1912     69.849998        70.683979

[739 rows x 2 columns]
```

In [31]:
```
Df.head()
```

Out[31]:

	Actual price	Predicted Price
1749	71.463997	73.091481
643	6.876000	6.916000
118	5.920000	5.883500
252	5.622000	5.612200
1311	50.638000	50.414980

Decision Tree for Stock Price Prediction

In [ ]:

In [32]:
```
from sklearn.tree import DecisionTreeRegressor

reg = DecisionTreeRegressor(max_depth=6)
reg.fit(x_train, y_train)
```

Out[32]:
```
DecisionTreeRegressor(max_depth=6)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [33]:
```
print("Train Accuracy :", reg.score(x_train, y_train))
print("Test Accuracy :", reg.score(x_test, y_test))
```

```
Train Accuracy : 0.9996440928260358
Test Accuracy : 0.9992422998685706
```

In [34]:
```
predicted=reg.predict(x_test)
```

In [35]:
```
print(x_test)
```

```
            Open        High         Low    Volume
1749   74.884003   75.374001   70.959999  86307000
643     6.832000    6.970000    6.784000   7183500
118     5.734000    5.994000    5.706000   3714500
252     5.558000    5.650000    5.534000   4446000
1311   50.220001   50.849998   49.933998  14454500
...          ...         ...         ...       ...
794    31.400000   32.459999   31.000000  64659500
2429  167.800003  172.699997  164.440002  75961000
517     7.000000    7.042000    6.476000  12846500
1943   63.299999   64.150002   61.933998  37421500
1912   70.199997   71.931999   69.725998  20988500

[739 rows x 4 columns]
```

In [36]:
```
print(Df)
```

```
      Actual price  Predicted Price
1749     71.463997        73.091481
643       6.876000         6.916000
118       5.920000         5.883500
252       5.622000         5.612200
1311     50.638000        50.414980
...            ...              ...
794      32.368000        31.765420
2429    166.757996       167.381917
517       6.670000         6.815400
1943     62.712002        62.814939
1912     69.849998        70.683979

[739 rows x 2 columns]
```

In [37]:
```
Df.head(10)
```

Out[37]:

	Actual price	Predicted Price
1749	71.463997	73.091481
643	6.876000	6.916000
118	5.920000	5.883500
252	5.622000	5.612200
1311	50.638000	50.414980
1083	45.270000	45.671620
719	11.158000	11.356760
1467	50.293999	50.509181
2407	113.912003	113.256839
1592	40.551998	40.329599

In [ ]:

In [ ]:

In [ ]: