stock price prediction

Stock Price Prediction using machine learning algorithm helps you discover the future value of company stock and other financial assets traded on an exchange. The entire idea of predicting stock prices is to gain significant profits. Predicting how the stock market will perform is a hard task to do. There are other factors involved in the prediction, such as physical and psychological factors, rational and irrational behavior, and so on. All these factors combine to make share prices dynamic and volatile. This makes it very difficult to predict stock prices with high accuracy.

Dataset

the dataset you provide contains information on stock price data.Each row represents the trading information for a specific date.

1.Open :- the price which stocks as started a trading when the market open on a particular deal ...

2.Close :- the price of a individual stock when the market close on a particular deal

3.High :- the high column is the highest price at which stock price during the period.

4.Low :- the low column is the lowest price at which stock price during the period

5.Volume :- volume is the amount at the total activity of trading during the period of the time.

6.Adj close :- adjustment closing is the calculation adjustment to made to the stock closing price .

Objective:

The objective of stock price prediction is to accurately forecast future price movements using historical data, financial indicators, and external factors like market sentiment and economic conditions. The model can focus on short-term, medium-term, or long-term trends, and its success is evaluated using metrics like MSE for regression or accuracy for classification. The model should also account for market conditions, risk management, and transaction costs to ensure practical, real-world applicability.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import metrics

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

In [2]:
```
tesla = pd.read_csv('tesla.csv')
tesla.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	29-06-2010	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	30-06-2010	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	01-07-2010	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	02-07-2010	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	06-07-2010	20.000000	20.00	15.830000	16.110001	16.110001	6866900

Correlation

In [29]:
```
tesla.corr()
```

Out[29]:

	Date	Open	High	Low	Close	Adj Close	Volume
Date	1.000000	0.929668	0.930309	0.929273	0.929903	0.929903	0.424967
Open	0.929668	1.000000	0.999578	0.999566	0.999054	0.999054	0.457938
High	0.930309	0.999578	1.000000	0.999490	0.999631	0.999631	0.466999
Low	0.929273	0.999566	0.999490	1.000000	0.999580	0.999580	0.448387
Close	0.929903	0.999054	0.999631	0.999580	1.000000	1.000000	0.458157
Adj Close	0.929903	0.999054	0.999631	0.999580	1.000000	1.000000	0.458157
Volume	0.424967	0.457938	0.466999	0.448387	0.458157	0.458157	1.000000

In [5]:
```
tesla.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2193 entries, 0 to 2192
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       2193 non-null   object 
 1   Open       2193 non-null   float64
 2   High       2193 non-null   float64
 3   Low        2193 non-null   float64
 4   Close      2193 non-null   float64
 5   Adj Close  2193 non-null   float64
 6   Volume     2193 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 120.1+ KB
```

In [6]:
```
# Assuming 'tesla' is a pandas DataFrame with a 'Date' column of type date
# Ensure 'Date' is in datetime format
tesla['Date'] = pd.to_datetime(tesla['Date'])
```

In [7]:
```
# Print the minimum and maximum dates
print(f'Dataframe contains stock prices between {tesla["Date"].min()} and 
print(f'Total days = {(tesla["Date"].max() - tesla["Date"].min()).days} da

```

```
Dataframe contains stock prices between 2010-06-29 00:00:00 and 2019-03-15 0
0:00:00
Total days = 3181 days
```

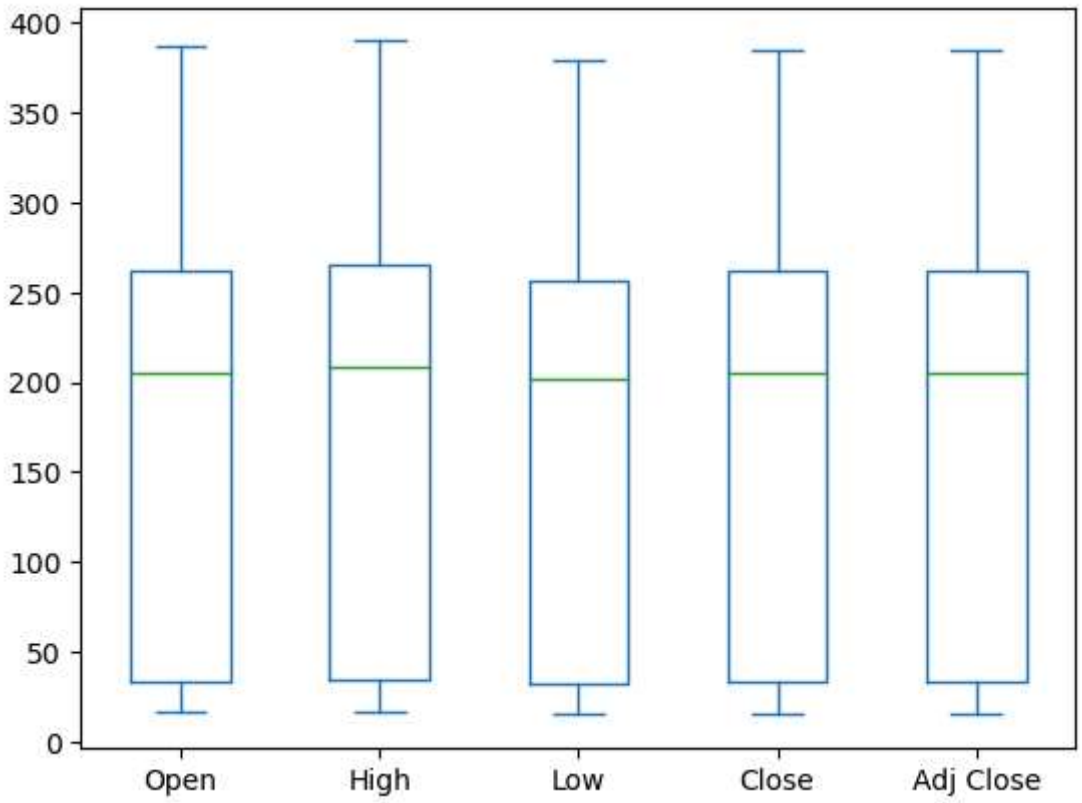
In [8]:
```
tesla.describe()
```

Out[8]:

	Date	Open	High	Low	Close	Adj Close	
count	2193	2193.000000	2193.000000	2193.000000	2193.000000	2193.000000	2.1
mean	2014-11-04 14:37:15.841313024	175.652882	178.710262	172.412075	175.648555	175.648555	5.0
min	2010-06-29 00:00:00	16.139999	16.629999	14.980000	15.800000	15.800000	1.1
25%	2012-08-29 00:00:00	33.110001	33.910000	32.459999	33.160000	33.160000	1.5
50%	2014-11-04 00:00:00	204.990005	208.160004	201.669998	204.990005	204.990005	4.1
75%	2017-01-09 00:00:00	262.000000	265.329987	256.209991	261.739990	261.739990	6.8
max	2019-03-15 00:00:00	386.690002	389.609985	379.350006	385.000000	385.000000	3.7
std	NaN	115.580903	117.370092	113.654794	115.580771	115.580771	4.5

In [9]:
```
1 tesla[['Open','High','Low','Close','Adj Close']].plot(kind='box')
```

Out[9]: <Axes: >



In [10]:
```
import plotly.graph_objects as go

# Setting the layout for the plot
layout = go.Layout(
    title='Stock Prices of Tesla',
    xaxis=dict(
        title='Date',
        titlefont=dict(
            family='Courier New, monospace',
            size=18,
            color='#7f7f7f'
        )
    ),
    yaxis=dict(
        title='Price',
        titlefont=dict(
            family='Courier New, monospace',
            size=18,
            color='#7f7f7f'
        )
    )
)

# Assuming tesla is a DataFrame with 'Date' and 'Close' columns
# Correct data format with go.Scatter for plotting
tesla_data = [go.Scatter(x=tesla['Date'], y=tesla['Close'], mode='lines',


```

In [11]:
```
# Creating the figure
plot = go.Figure(data=tesla_data, layout=layout)
# Show the plot
plot.show()
```

In [12]:
```
# Building the regression model
from sklearn.model_selection import train_test_split

#For preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

#For model evaluation
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
```

In [13]:
```
#Split the data into train and test sets
X = np.array(tesla.index).reshape(-1,1)
Y = tesla['Close']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, r
```

In [14]:
```
# Feature scaling
scaler = StandardScaler().fit(X_train)
```

In [15]:
```
from sklearn.linear_model import LinearRegression
```

In [16]:
```
#Creating a linear model
lm = LinearRegression()
lm.fit(X_train, Y_train)
```

Out[16]:
```
LinearRegression()
```
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.

Train and Test

In [17]:
```
import plotly.graph_objs as go

# Assuming X_train and Y_train are defined and lm is your fitted model

# Create traces for actual and predicted values
trace0 = go.Scatter(
    x=X_train[:, 0],  # Assuming X_train is a 2D array (n_samples, n_featu
    y=Y_train,
    mode='markers',
    name='Actual'
)

trace1 = go.Scatter(
    x=X_train[:, 0],  # Using the same x values
    y=lm.predict(X_train),
    mode='lines',
    name='Predicted'
)

# Combine traces
tesla_data = [trace0, trace1]

# Define layout
layout = go.Layout(
    title='Actual vs Predicted Values',
    xaxis=dict(title='Day'),
    yaxis=dict(title='Values'),
    legend=dict(x=0, y=1)
)


```

In [18]:
```
# Create figure
plot2 = go.Figure(data=tesla_data, layout=layout)

# Show the plot
plot2.show()

```

Model Evaluation

from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

In [19]:
```
Y_pred = lm.predict(X_test)
```

In [20]:
```
Y_test.values
```

Out[20]:
```
array([254.839996, 206.550003, 233.389999, 310.700012, 122.269997,
        28.34    ,  29.129999, 169.619995, 231.960007, 232.740005,
       307.540009,  33.59    ,  29.809999, 249.240005,  27.860001,
       227.070007, 227.479996, 256.290009, 262.019989,  19.1     ,
       284.179993, 362.220001,  94.470001, 277.390015, 256.880005,
       219.529999, 219.990005,  22.73    , 228.889999, 243.179993,
        30.66    , 230.009995, 243.149994,  20.549999,  32.07    ,
       231.770004, 314.070007, 232.339996, 307.019989, 144.699997,
       232.5     , 248.589996,  29.280001, 218.339996,  20.709999,
       281.190002, 236.610001, 218.960007, 253.979996, 231.550003,
        33.810001, 250.479996, 179.      , 207.      , 104.949997,
       122.43    , 200.419998, 377.640015,  30.129999,  26.870001,
       236.800003, 225.710007, 343.399994,  27.66    , 199.100006,
       220.009995, 305.640015, 280.309998, 343.450012, 298.329987,
       152.440002,  30.879999, 249.990005, 252.949997,  97.349998,
       243.690002,  51.009998, 147.860001, 120.25    ,  24.950001,
        28.959999, 196.559998, 376.399994, 182.839996, 355.899994,
        26.1     ,  20.540001,  31.23    , 257.920013, 139.649994,
        20.940001, 248.839996,  23.73    ,  26.219999, 246.210007,
```

MSE

In [21]:
```
'MSE'.ljust(10)
mse=(Y_train, lm.predict(X_train))
```

In [22]:
```
print("MSE:",mse)
```

```
MSE: (365      34.189999
1111    248.089996
1581    196.610001
1990    277.850006
1753    375.339996
           ...
599      31.610001
1599    188.020004
1361    217.750000
1547    225.000000
863     124.169998
Name: Close, Length: 1535, dtype: float64, array([ 50.40381287, 177.54354106,
257.64497839, ..., 220.15068857,
       251.85040633, 135.27725072]))
```

MAE

In [23]:
```
mae=(Y_test,Y_pred)

```

In [24]:
```
print("mae:",mae)
```

```
1063    263.820007
543      29.950001
Name: Close, Length: 658, dtype: float64, array([145.84382331, 184.3606846
6, 223.04797461, 342.51841624,
       118.06396313,  44.43881222,  31.48623937, 141.58310855,
       214.5265451 , 233.61454719, 306.72841233,  65.23110021,
        70.34395791, 271.108837  ,  29.10023911, 222.19583166,
       172.94196913, 200.04011495, 202.5965438 ,  -5.49676467,
       326.15727159, 330.41798635, 114.65539132, 167.82911142,
       343.71141637, 267.01855084, 252.70254928,  19.38580948,
       148.05939498, 197.14282892,  74.77510125, 249.46440607,
       208.22068727,  -8.56447929,  48.69952697, 218.10554549,
       281.33455241, 242.98811964, 356.66398922, 132.89125046,
       159.13725333, 275.02869458,  85.17124524, 233.10326142,
        -3.281193  , 167.48825424, 196.63154315, 242.47683387,
       176.52096952, 195.09768584,  94.03353193, 274.5174088 ,
       230.71726116, 179.58868414, 113.2919626 , 119.42739185,
       254.91812095, 297.69569705,  28.24809616,  24.49866718,
       196.46111456, 181.29297004, 297.01398269,  23.98738141,
       258.83797852, 219.98025998, 358.7091323 , 168.51082579,
```

R2S

In [25]:
```
'r2s'.ljust(10)
r2s=(Y_train, lm.predict(X_train))
```

In [26]:
```
1 print("r2s:",r2s)
```

```
r2s: (365      34.189999
1111    248.089996
1581    196.610001
1990    277.850006
1753    375.339996
           ...
599      31.610001
1599    188.020004
1361    217.750000
1547    225.000000
863     124.169998
Name: Close, Length: 1535, dtype: float64, array([ 50.40381287, 177.54354106,
257.64497839, ..., 220.15068857,
       251.85040633, 135.27725072]))
```

In [27]:
```
1 rmse=mse
2 data_rmse= {'Actual (Y_test)':Y_test, 'Predicted (Y_pred)':Y_pred}
3 df_rmse = pd.DataFrame(data_rmse)
4 df_rmse.head()
```

Out[27]:

	Actual (Y_test)	Predicted (Y_pred)
925	254.839996	145.843823
1151	206.550003	184.360685
1378	233.389999	223.047975
2079	310.700012	342.518416
762	122.269997	118.063963

In [28]:
```
1 print("rmse:",rmse)
```

```
rmse: (365      34.189999
1111    248.089996
1581    196.610001
1990    277.850006
1753    375.339996
           ...
599      31.610001
1599    188.020004
1361    217.750000
1547    225.000000
863     124.169998
Name: Close, Length: 1535, dtype: float64, array([ 50.40381287, 177.54354106,
257.64497839, ..., 220.15068857,
       251.85040633, 135.27725072]))
```

In [ ]:
```
1
```