Linear Regression

Linear regression is a data analysis technique that predicts the value of unknown data by using another related and known data value. It mathematically models the unknown or dependent variable and the known or independent variable as a linear equation.

The mathematical formula of the linear regression can be written as **y = b0 + b1*x + e**, where:

b0 and b1 are known as the regression beta coefficients or parameters: b0 is the intercept of the regression line; that is the predicted value when x = 0. b1 is the slope of the regression line. e is the error term (also known as the residual errors), the part of y that can be explained by the regression model

Linear Regression

It's all about understanding and quantifying relationships. Here are the key components:

- Dependent Variable: The outcome we want to predict.
- Independent Variables: The factors that influence the dependent variable.
- Assumptions: Linear relationship, multivariate normality, little multicollinearity, no auto-correlation, and homoscedasticity.

Real-world Applications:

- Predictive Modeling: We use Linear Regression to make predictions, like forecasting sales, stock prices, or even real estate values.

- Hypothesis Testing: It's not just about prediction; we can test hypotheses and understand the relationships between variables.

- Quality Control: In manufacturing, Linear Regression helps us analyze the connection between factors and product quality.

- Economic Analysis: Economists use it to examine the impact of variables like inflation and interest rates on economic indicators.

Challenges and Caution:

Linear Regression is a valuable tool, but it's not without its challenges. Some points to keep in mind:

- Assumption Validity: Check and address the assumptions for reliable results.
- Overfitting: Avoid using too many independent variables; overfitting can be a pitfall.
- Outliers: Identify and address outliers that can skew your analysis.
- Causation vs. Correlation: Linear Regression shows relationships, not causation. Be cautious when interpreting results.

In [1]:
```
# pip install chart-studio
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import plot

#for offline plotting
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected = True)
```

In [3]:
```
# Load Tesla stock price data from a CSV file
tesla = pd.read_csv('tesla.csv')
```

In [4]:
```
# Display the first few rows of the dataset
tesla.head()
```

Out[4]:

	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

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]:
```
# Convert the 'Date' column to a datetime data type
tesla['Date'] = pd.to_datetime(tesla['Date'])
```

```
C:\Users\Garima\AppData\Local\Temp\ipykernel_7236\1941679751.py:2: UserWarning:

Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was specified. This may lead to inconsistently parsed dates! Spec
ify a format to ensure consistent parsing.
```

Exploratory Data Analysis

In [7]:
```
# Print the date range in the dataset
print(f'Dataframe contains stock prices between {tesla.Date.min()} to {tesla.Date.max()}')

# Calculate and print the total number of days in the dataset
print(f'Total days = {(tesla.Date.max()- tesla.Date.min()).days} days')
```

```
Dataframe contains stock prices between 2010-01-07 00:00:00 to 2019-12-03 00:00:00
Total days = 3617 days
```

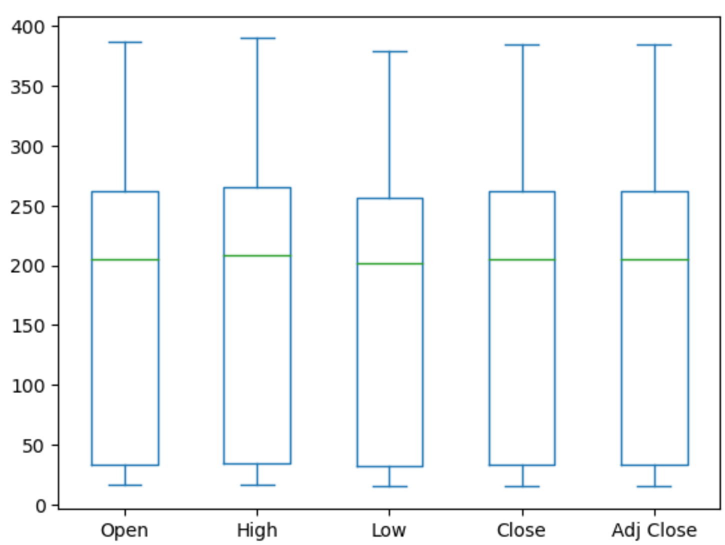
In [8]:
```
# Display summary statistics for numerical columns
tesla.describe()
```

Out[8]:

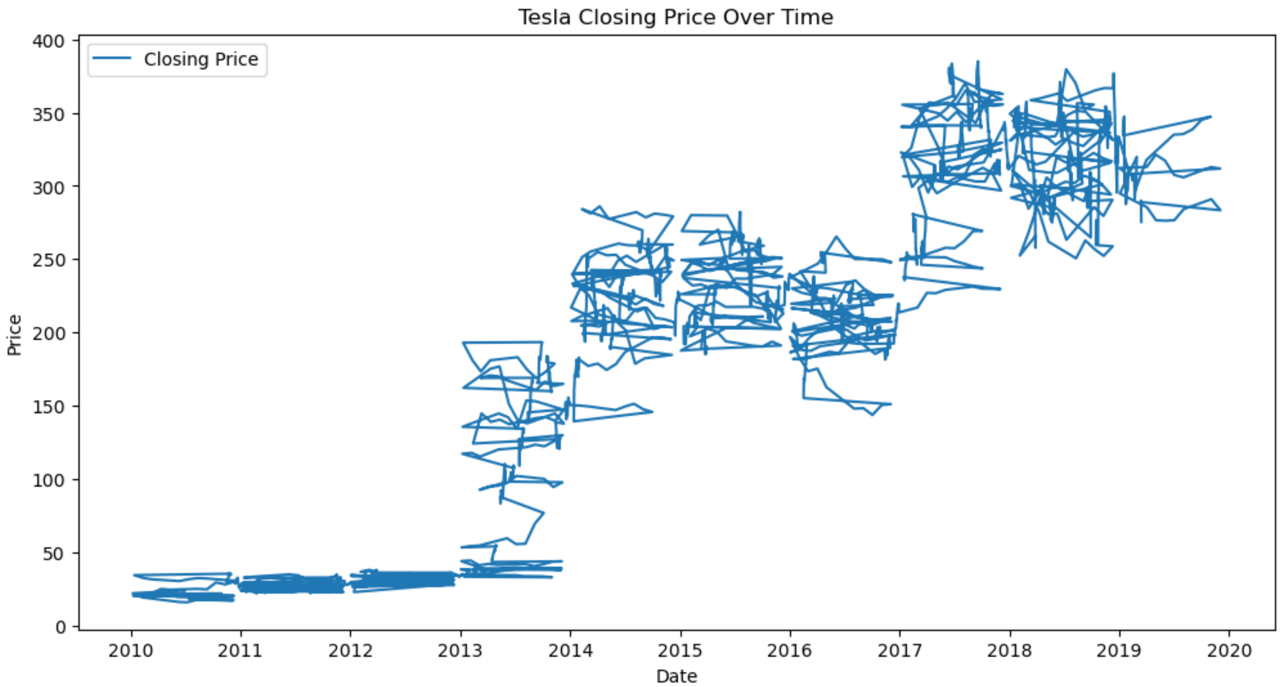
	Open	High	Low	Close	Adj Close	Volume
count	2193.000000	2193.000000	2193.000000	2193.000000	2193.000000	2.193000e+03
mean	175.652882	178.710262	172.412075	175.648555	175.648555	5.077449e+06
std	115.580903	117.370092	113.654794	115.580771	115.580771	4.545398e+06
min	16.139999	16.629999	14.980000	15.800000	15.800000	1.185000e+05
25%	33.110001	33.910000	32.459999	33.160000	33.160000	1.577800e+06
50%	204.990005	208.160004	201.669998	204.990005	204.990005	4.171700e+06
75%	262.000000	265.329987	256.209991	261.739990	261.739990	6.885600e+06
max	386.690002	389.609985	379.350006	385.000000	385.000000	3.716390e+07

In [9]:
```
# Create a box plot to visualize the distribution of selected price columns
tesla[['Open','High','Low','Close','Adj Close']].plot(kind = 'box')
```

Out[9]: <Axes: >



In [10]:
```
plt.figure(figsize=(12, 6))
plt.plot(tesla['Date'], tesla['Close'], label='Closing Price')
plt.title('Tesla Closing Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```



In [11]:
```
# Setting the layout for our 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'
        )
    )
)
# Create a Plotly plot for Tesla's closing prices
tesla_data = [{'x':tesla['Date'], 'y':tesla['Close']}]
plot = go.Figure(data = tesla_data, layout = layout)
```

In [12]:
```
#iplot(plot) #plotting offline
iplot(plot)
```

Stock Prices of Tesla

In [13]:
```python
# Calculate the correlation matrix for the dataset
correlation_matrix = tesla.corr()
print(correlation_matrix)
```

```
               Open      High       Low     Close  Adj Close    Volume
Open       1.000000  0.999578  0.999566  0.999054   0.999054  0.457938
High       0.999578  1.000000  0.999490  0.999631   0.999631  0.466999
Low        0.999566  0.999490  1.000000  0.999580   0.999580  0.448387
Close      0.999054  0.999631  0.999580  1.000000   1.000000  0.458157
Adj Close  0.999054  0.999631  0.999580  1.000000   1.000000  0.458157
Volume     0.457938  0.466999  0.448387  0.458157   0.458157  1.000000
```

```
C:\Users\Garima\AppData\Local\Temp\ipykernel_7236\4183763580.py:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid co
lumns or specify the value of numeric_only to silence this warning.
```

In [14]:
```python
# 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 [15]:
```python
#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, random_state = 101)
```

In [16]:
```python
# Feature scaling
scalar = StandardScaler().fit(X_train)
```

In [17]:
```python
from sklearn.linear_model import LinearRegression
```

In [18]:
```python
#Creating a Linear model
lm = LinearRegression()
lm.fit(X_train, Y_train)
```

Out[18]:
```
▾ LinearRegression
LinearRegression()
```

In [19]:
```python
#Plot actual and predicted values for train dataset
trace0 = go.Scatter(
    x = X_train.T[0],
    y = Y_train,
```

```
    mode = 'markers',
    name = 'Actual'
)
trace1 = go.Scatter(
x = X_train.T[0],
y = lm.predict(X_train).T,
mode = 'lines',
name = 'Predicted'
)
tesla_data = [trace0,trace1]
layout.xaxis.title.text = 'Day'
plot2 = go.Figure(data = tesla_data, layout = layout)
```

In [20]:
```
iplot(plot2)
```

In [21]:
```
#Calculate scores for model evaluation
scores = f'''
{'Metric'.ljust(10)}{'Train'.center(20)}{'Test'.center(20)}
{'r2_score'.ljust(10)}{r2_score(Y_train, lm.predict(X_train))}\t{r2_score(Y_test, lm.predict(X_test))}
{'MSE'.ljust(10)}{mse(Y_train, lm.predict(X_train))}\t{mse(Y_test, lm.predict(X_test))}
'''
print(scores)
```

```
Metric            Train                Test
r2_score  0.8658871776828707    0.8610649253244574
MSE       1821.3833862936174    1780.987539418845
```

In [ ]: