Multiple Linear Regression

➢ It is a statistical method used to study the linear relationship between a dependent variable and multiple independent variables.

➢ MLR, which involves more than one independent variable.

➢ Multiple linear regression is an extension of simple linear regression,where multiple independent variables are used to predict the dependent variable.

➢ Scikit-learn, a machine learning library in Python, can be used to implement multiple linear regression models and to read, preprocess, and split data.

➢ Categorical variables can be handled in multiple linear regression using one-hot encoding or label encoding.

Multiple Linear Regression (MLR) is basically indicating that we will have many features Such as f1, f2, f3, f4, and our output feature f5. If we take the same example as above we discussed, suppose:

f1 is the size of the house,

f2 is bad rooms in the house,

f3 is the locality of the house,

f4 is the condition of the house, and

f5 is our output feature, which is the price of the house.

Y=A+B1* 1+B2* 2+B3* 3+B4* 4

Train a Model for Multiple Linear Regression

• Step 1: Reading the Dataset

• Step 2: Handling Categorical Variables

• Step 3: Splitting the Data

• Step 4: Applying the Model

Student Performance Analysis using Multiple Linear Regression

Import Important Libraries :

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

Import Dataset :

In [2]:
```
df = pd.read_csv('Student_Performance.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0
2	8	51	Yes	7	2	45.0
3	5	52	Yes	5	2	36.0
4	7	75	No	8	5	66.0

In [4]:
```
df.tail()
```

Out[4]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
9995	1	49	Yes	4	2	23.0
9996	7	64	Yes	8	5	58.0
9997	6	83	Yes	8	5	74.0
9998	9	97	Yes	7	0	95.0
9999	7	74	No	8	1	64.0

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Hours Studied                     10000 non-null  int64  
 1   Previous Scores                   10000 non-null  int64  
 2   Extracurricular Activities        10000 non-null  object 
 3   Sleep Hours                       10000 non-null  int64  
 4   Sample Question Papers Practiced  10000 non-null  int64  
 5   Performance Index                 10000 non-null  float64
dtypes: float64(1), int64(4), object(1)
memory usage: 468.9+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	4.992900	69.445700	6.530600	4.583300	55.224800
std	2.589309	17.343152	1.695863	2.867348	19.212558
min	1.000000	40.000000	4.000000	0.000000	10.000000
25%	3.000000	54.000000	5.000000	2.000000	40.000000
50%	5.000000	69.000000	7.000000	5.000000	55.000000
75%	7.000000	85.000000	8.000000	7.000000	71.000000
max	9.000000	99.000000	9.000000	9.000000	100.000000

EDA :

In [7]:
```
df['Extracurricular Activities'].unique()
```

Out[7]:
```
array(['Yes', 'No'], dtype=object)
```

In [8]:
```
# Encoding categorical values

d = {'Yes':0, 'No':1}
df['Extracurricular Activities'] = df['Extracurricular Activities'].map(d)
```

In [9]:
```
df.head()
```

Out[9]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	0	9	1	91.0
1	4	82	1	4	2	65.0
2	8	51	0	7	2	45.0
3	5	52	0	5	2	36.0
4	7	75	1	8	5	66.0

In [10]:
```
df.tail()
```

Out[10]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
9995	1	49	0	4	2	23.0
9996	7	64	0	8	5	58.0
9997	6	83	0	8	5	74.0
9998	9	97	0	7	0	95.0
9999	7	74	1	8	1	64.0

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Hours Studied                     10000 non-null  int64  
 1   Previous Scores                   10000 non-null  int64  
 2   Extracurricular Activities        10000 non-null  int64  
 3   Sleep Hours                       10000 non-null  int64  
 4   Sample Question Papers Practiced  10000 non-null  int64  
 5   Performance Index                 10000 non-null  float64
dtypes: float64(1), int64(5)
memory usage: 468.9 KB
```

Splitting the Dataset :

In [12]:
```
X = df.drop('Performance Index', axis=1)
Y = df['Performance Index']
```

In [13]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.2, random_state=42)
```

Training the Model :

In [14]:
```
from sklearn.linear_model import LinearRegression
```

In [15]:
```
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

Out[15]:

▾ LinearRegression (https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.LinearRegression.html)

LinearRegression()

Evaluating Results :

In [16]:
```
y_pred = regressor.predict(x_test)
```

In [17]:
```
out = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
out
```

Out[17]:

	Actual	Predicted
6252	51.0	54.711854
4684	20.0	22.615513
1731	46.0	47.903145
4742	28.0	31.289767
4521	41.0	43.004570
...	...	...
6412	45.0	46.886280
8285	66.0	62.698025
7853	16.0	16.793420
1095	65.0	63.343274
6929	47.0	45.942623

2000 rows × 2 columns

Model Evaluation :

In [18]:
```
from sklearn.metrics import r2_score, mean_squared_error
```

In [19]:
```
r2 = r2_score(y_test, y_pred)
r2

# r2_score = 0.9889 ~ 1
# So, the model has good fit to the dataset
```

Out[19]:
```
0.9889832909573145
```

In [20]:
```
sq_err = mean_squared_error(y_test, y_pred)
sq_err
```

Out[20]:
```
4.082628398521858
```

In [21]:
```
regressor.coef_
```

Out[21]:
```
array([ 2.85248393,  1.0169882 , -0.60861668,  0.47694148,  0.19183144])
```

- So, the equation for multiple linear regression can be written as follows:

Performance Index' = 2.85*'Hours Studied' + 1.01*'Previous Scores' + (-0.60)*'Extracurricular Activities'* + *0.47*'Sleep Hours' + 0.19*'Sample Question Papers Practiced'

In [22]:
```
regressor.intercept_
```

Out[22]:
```
-33.31332953597987
```

Red Wine Quality

In [23]:
```
data = pd.read_csv('winequality-red.csv')
print(data.head())
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

In [24]:
```
data.describe()
```

Out[24]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	der
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003

In [25]:
```
data.isnull().sum()
```

Out[25]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Prepare Data:

In [26]:
```
X = data[['fixed acidity', 'volatile acidity', 'alcohol']].values
y = data['quality'].values
```

Split Data into Training and Testing Sets:

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0
```

Build and Train the Multiple Linear Regression Model:

In [28]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[28]:
```
LinearRegression
(https://scikit-learn.org/1.4/modules/generated/sklearn.linear_model.LinearRegression.html)
LinearRegression()
```

Make Predictions:

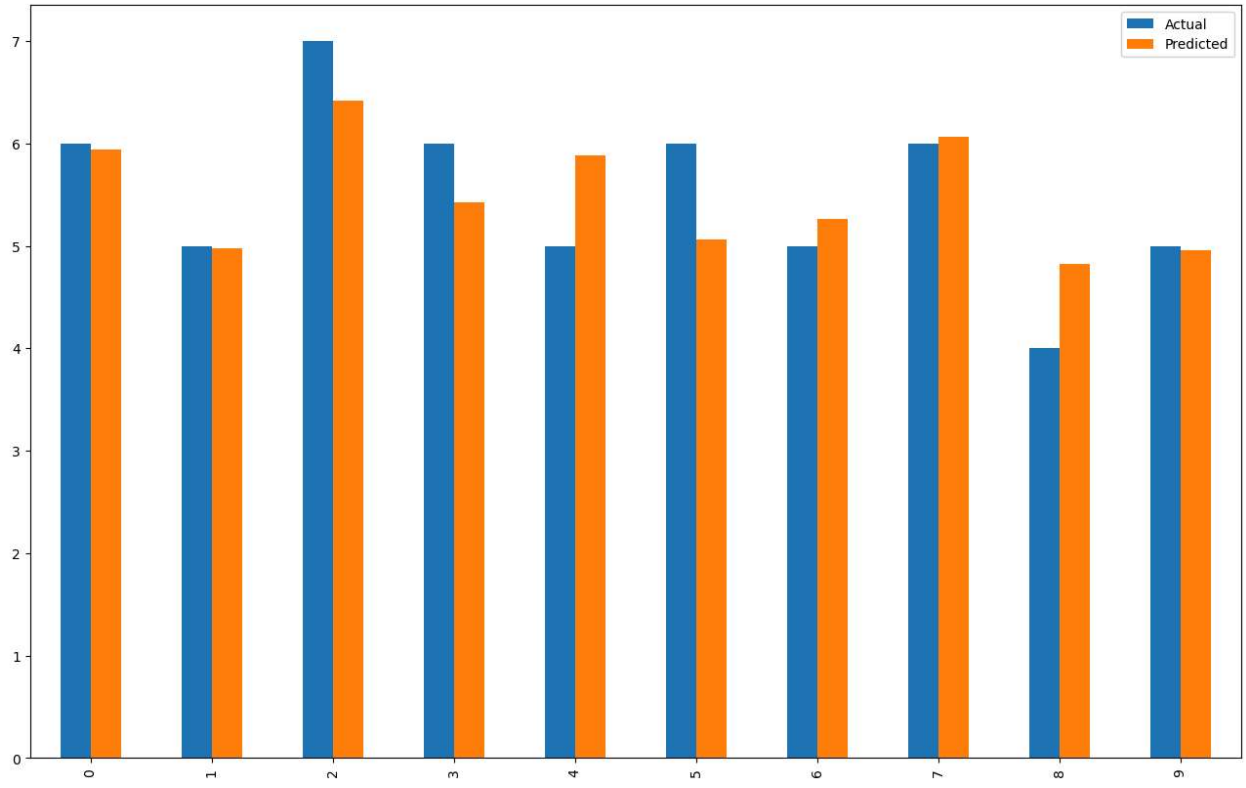
In [29]:
```
y_pred = regressor.predict(X_test)
```

In [30]:
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df1 = df.head(10)
df1
```

Out[30]:

	Actual	Predicted
0	6	5.940980
1	5	4.981045
2	7	6.417707
3	6	5.422189
4	5	5.886753
5	6	5.063754
6	5	5.263226
7	6	6.068822
8	4	4.823850
9	5	4.962100

In [31]:
```
df1.plot(kind='bar',figsize=(16,10))
plt.show()
```



Evaluate the Model:

In [32]:
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
```

```
Mean Absolute Error: 0.4879795661109293
Mean Squared Error: 0.4096570425100602
```

Interpret the Results:

In [33]:
```
coefficients = regressor.coef_
intercept = regressor.intercept_

print("Coefficients:", coefficients)
print("Intercept:", intercept)
```

```
Coefficients: [ 0.03543676 -1.34814989  0.32700295]
Intercept: 2.660261640127603
```

In [ ]: