Isha Hanmante

Data Science & Business Analytics Intern (Batch - June2021)

Task 1: Prediction using Supervised ML

---Linear Regression with Python Scikit Learn---

In this section we will see how the Python Scikit-Learn library for machine learning can be used to implement regression functions. We will start with simple linear regression involving two variables.

----Problem statement---

In this regression task we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied. This is a simple linear regression task as it involves just two variables.

--To Predict--

What will be predicted score if a student studies for 9.25 hrs/ day?

Importing Libraries

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

Loading Dataset

In [2]:
```
url= "http://bit.ly/w-data "
data=pd.read_csv(url)
df=data
print("****** Data imported successfully! ******")
data #Displaying the data
```

```
****** Data imported successfully! ******
```

Out[2]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

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

Out[3]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

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

Out[4]:

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

Checking Null Values

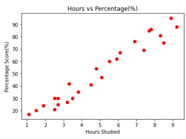
In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Hours     0
Scores    0
dtype: int64
```

No Null values found, so no need to clean this data

Plotting the distribution of scores

In [18]:
```
plt.scatter(df['Hours'], df['Scores'], color = 'red')
plt.title('Hours vs Percentage(%)')
plt.xlabel('Hours Studied')
plt.ylabel("Percentage Score(%)")
plt.show()
```



Linear Regression model - Preparing the data and splitting it in testing

In [7]:
```
# Dividing the data into attributes and labels.
x = df.iloc[:, :-1].values
y = df.iloc[:, 1].values
```

Splitting the data into Training and Testing Sets

In [8]:
```
# Using in-built method of sci-kit learn of train_test_split()
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0
)
```

Training the model

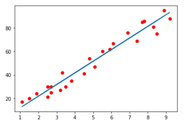
In [9]:
```
from sklearn.linear_model import LinearRegression
```

In [10]:
```
model = LinearRegression()
model.fit(x_train, y_train)
print("Model Trained!")
```

```
Model Trained!
```

Plotting the Regression Line

In [11]:
```
# Plotting the regression line # formula for line is y=m*x + c
line = model.coef_*x + model.intercept_
plt.scatter(x, y, color = 'red')
plt.plot(x, line)
plt.show()
```



Making Predictions

In [12]:
```
#Predicting scores for model
print(x_test)
y_pred = model.predict(x_test)
```

```
[[1.5]
 [3.2]
 [7.4]
 [2.5]
 [5.9]
 [3.8]
 [1.9]]
```

Comparing Actual vs Predicted

In [13]:
```
df1 = pd.DataFrame({'Actual' : y_test, 'Predicted' : y_pred})
df1
```

Out[13]:

	Actual	Predicted
0	20	16.844722
1	27	33.745575
2	69	75.500624
3	30	26.786400
4	62	60.588106
5	35	39.710582
6	24	20.821393

In [14]:
```
#Checking the accuracy of training and test scores
print('Test Score')
print(model.score(x_test, y_test))
print('Training Score')
print(model.score(x_train, y_train))
```

```
Test Score
0.9367661043365055
Training Score
0.9484509249328872
```

Testing with custom data

In [15]:
```
hrs = [[9.25]]
predict = model.predict(hrs)
print("No. of Hours = {}".format(hrs))
print("Predicted Score = {}".format(predict[0]))
```

```
No. of Hours = [[9.25]]
Predicted Score = 93.89272889341655
```

Evaluating the Model

In [16]:
```
from sklearn import metrics
print('Mean Absolute Error(MAE) :', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error(MSE) :', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error(RMSE) :', np.sqrt(metrics.mean_squared_error(y_test, y_pred
)))
```

```
Mean Absolute Error(MAE) : 4.130879918502486
Mean Squared Error(MSE) : 20.33292367497997
Root Mean Squared Error(RMSE) : 4.509204328368055
```

In [ ]:
```

```