MAHATHI SRISOWMYA

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 function.we will start with simple linear regression involving two variables

Simple Linear Regression

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 invovlves just two variables.

Importing Dataset

In [1]:
```
######## https://raw.githubusercontent.com/AdiPersonalWorks/Random/master/student_scores%20-%20student_scores.csv
```

In [2]:
```
# Importing all libraries required in this notebook
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import warnings as wg
wg.filterwarnings("ignore")
```

In [3]:
```
# data extraction of url
score_data = pd.read_csv("http://bit.ly/w-data")
score_data.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]:
```
score_data.tail()
```

Out[4]:

	Hours	Scores
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

In [5]:
```
score_data.describe()
```

Out[5]:

	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

In [6]:
```
score_data.shape
```

Out[6]:
```
(25, 2)
```

In [7]:
```
score_data.isnull().sum()
```

Out[7]:
```
Hours     0
Scores    0
dtype: int64
```

Visualization of Data set

In [8]:
```
score_data.plot(x='Hours',y='Scores',style='o')
plt.xlabel('Number of Hours studied')
plt.ylabel('Scores obtained by students')
plt.title('Number of Hours vs Percentage')
plt.show()
```

Preparing the data

In [9]:
```
score_data.corr()
```

Out[9]:

	Hours	Scores
Hours	1.000000	0.976191
Scores	0.976191	1.000000

In [10]:
```
x = score_data.iloc[:, :-1].values
x
```

Out[10]:
```
array([[2.5],
       [5.1],
       [3.2],
       [8.5],
       [3.5],
       [1.5],
       [9.2],
       [5.5],
       [8.3],
       [2.7],
       [7.7],
       [5.9],
       [4.5],
       [3.3],
       [1.1],
       [8.9],
       [2.5],
       [1.9],
       [6.1],
       [7.4],
       [2.7],
       [4.8],
       [3.8],
       [6.9],
       [7.8]])
```

In [11]:
```
y = score_data.iloc[:, 1].values
y
```

Out[11]:
```
array([21, 47, 27, 75, 30, 20, 88, 60, 81, 25, 85, 62, 41, 42, 17, 95, 30,
       24, 67, 69, 30, 54, 35, 76, 86], dtype=int64)
```

In [12]:
```
#splitting the data trainning and testing
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=0)
```

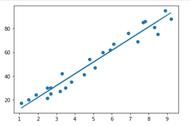
Training the Algorithm

We have split our data into training and testing sets, and now is finally the time to train our algorithm

In [13]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train, y_train)
print("Training complete")
```

```
Training complete
```

In [14]:
```
# Plotting the regression line
l = reg.coef_*x+reg.intercept_
plt.scatter(x,y)
plt.plot(x,l)
plt.show()
```



In [15]:
```
print("Intercept is : ", reg.intercept_)
print("Coefficient is:", reg.coef_)
```

```
Intercept is :  2.018160041434662
Coefficient is: [9.91065648]
```

Making Predictions

In [16]:
```
print(x_test) #Testing data - In hours
y_pred = reg.predict(x_test) #Predicting the scores
```

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

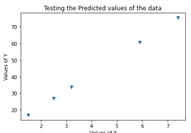
In [17]:
```
#Comparing actual vs Predicted values
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[17]:

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

In [18]:
```
plt.scatter(x_test,y_test)
plt.xlabel('VALUES OF X')
plt.ylabel('VALUES OF Y')
plt.title('Testing the actual values of the data')
plt.show()
```

In [19]:
```
plt.scatter(x_test,y_pred,marker ='v')
plt.xlabel('Values of X')
plt.ylabel('Values of Y')
plt.title('Testing the Predicted values of the data')
plt.show()
```



MODEL DIAGNOSIS

Mean absolute error

Mean Squared error

Root Mean Squared error

In [23]:
```
from sklearn import metrics
print ('Mean Absolute Error:',metrics.mean_absolute_error(y_test,y_pred))
```

```
Mean Absolute Error: 4.183859899002982
```

In [24]:
```
print('Mean Squared Error:',metrics.mean_squared_error(y_test,y_pred))
```

```
Mean Squared Error: 21.598769307217456
```

In [25]:
```
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

```
Root Mean Squared Error: 4.647447612100373
```

CONCLUSION: I have successfully predicted the percentage of marks by a student based on study hours