Author : Pamulapati Kirthana

GRIPJUNE21

DATA SCIENCE & BUSINESS ANALYTICS

Prediction using Supervised ML

In [ ]:

```

```

IMPORTING REQUIRED LIBRARIES

In [1]:

```
# Importing all libraries required for the task
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

READING THE DATA

In [2]:

```
# Reading data from remote link
url="https://raw.githubusercontent.com/AdiPersonalWorks/Random/master/student_scores%20-%20
df=pd.read_csv(url)
print("Data imported successfully")
df
```

```
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]:

```
#The method shape returns the Row and Column of the given dataset.
df.shape
```

Out[3]:

```
(25, 2)
```

The given dataset has two columns and twenty-five rows.

In [4]:

```
#Now,we will be using the method info() which will give the information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
```

In [5]:

```
#The method describe() calculates the statistical information (like mean,number of data pre
df.describe()
```

Out[5]:

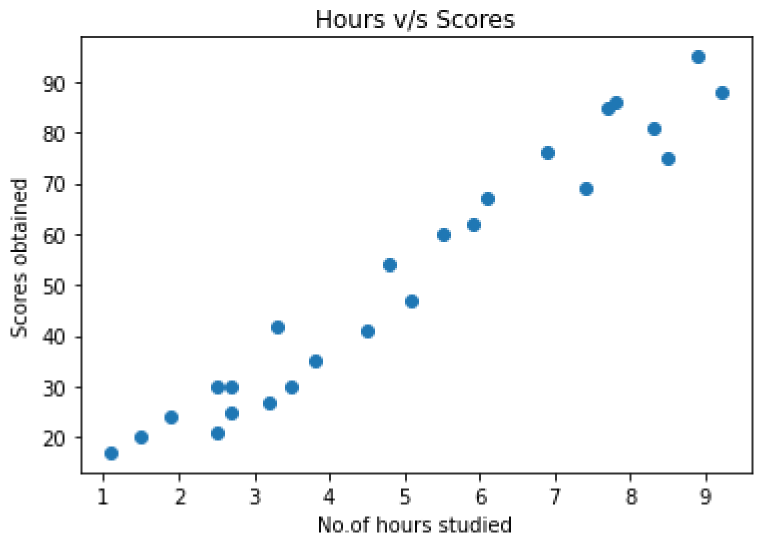
	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

INTERPRETING THE DATASET

Plotting the data points on 2-D graph to find the relationship between the variables.

In [6]:

```
# Plotting the distribution of scores
plt.scatter(x="Hours",y="Scores",data=df)
plt.title("Hours v/s Scores")
plt.xlabel("No.of hours studied")
plt.ylabel("Scores obtained")
plt.show()
```



From the graph above, we can clearly see that there is a positive linear relation between the number of hours studied and percentage of score.

PREPARING THE DATASET

The next step is to divide the data into "attributes" (inputs) and "labels" (outputs).

In [7]:

```
#variable 'x' contains the list of number of hours studied and variable 'y' contains the li
x = df.iloc[:, :-1].values
y = df.iloc[:, 1].values
```

In [8]:

```
#Displays list of number of hours studied
x
```

Out[8]:

```
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 [9]:

```
#Displays list of scores obtained.
y
```

Out[9]:

```
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)
```

Now,we have divided the data into attribute(independent variable) and labels(dependent variable).

Now that we have our attributes and labels, the next step is to split this data into training and testing sets. We'll do this by using Scikit-Learn's built-in train_test_split() method:

In [10]:

```
from sklearn.model_selection import train_test_split
# Splitting the data into training and testing data by 80:20 proportion.
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.2, random_state=0)
```

In [11]:

```
#x's training data
x_train
```

Out[11]:

```
array([[3.8],
       [1.9],
       [7.8],
       [6.9],
       [1.1],
       [5.1],
       [7.7],
       [3.3],
       [8.3],
       [9.2],
       [6.1],
       [3.5],
       [2.7],
       [5.5],
       [2.7],
       [8.5],
       [2.5],
       [4.8],
       [8.9],
       [4.5]])
```

In [12]:

```
#x's test data
x_test
```

Out[12]:

```
array([[1.5],
       [3.2],
       [7.4],
       [2.5],
       [5.9]])
```

In [13]:

```
#y's training data
y_train
```

Out[13]:

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

In [14]:

```
#y's test data
y_test
```

Out[14]:

```
array([20, 27, 69, 30, 62], dtype=int64)
```

Training the Algorithm

In [15]:

```
from sklearn.linear_model import LinearRegression
LRM = LinearRegression()
LRM.fit(x_train, y_train)

print("Training complete.")
```

```
Training complete.
```

Now, we have our training and test data.Our next step is to train our algorithm.

In [16]:

```
#The attribute 'coef_' computes the coefficients of the features in the decision function
LRM.coef_
```

Out[16]:

```
array([9.91065648])
```

In [17]:

```
#The attribute 'intercept_' is to represents the constant(bias), added to the decision func
LRM.intercept_
```

Out[17]:

```
2.018160041434683
```

Now that we computed coefficient and intercept, let's calculate the accuracy of the model.

In [18]:

```
#Checking the percentage of correct predictions for the test data.
print("Accuracy : ",LRM.score(x_test, y_test)*100)
```

```
Accuracy :  94.54906892105356
```

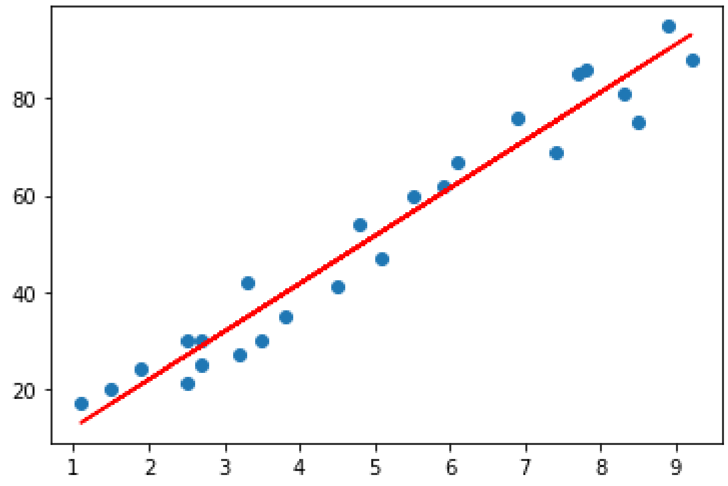
Plotting Regression Line

In [19]:

```
line = LRM.coef_*x+LRM.intercept_
```

In [20]:

```
plt.scatter(x, y)
plt.plot(x, line,color='red');
plt.show()
```



Making Prediction

In [21]:

```
#Predicting the scores using our trained algorithm
#The predict() method will predict the label of a new set of data(given a trained model).
y_prediction = LRM.predict(x_test)
y_prediction
```

Out[21]:

```
array([16.88414476, 33.73226078, 75.357018  , 26.79480124, 60.49103328])
```

Here,we calculated our predictied scores using our trained model.

Let's compare actual and predicted data.

In [22]:

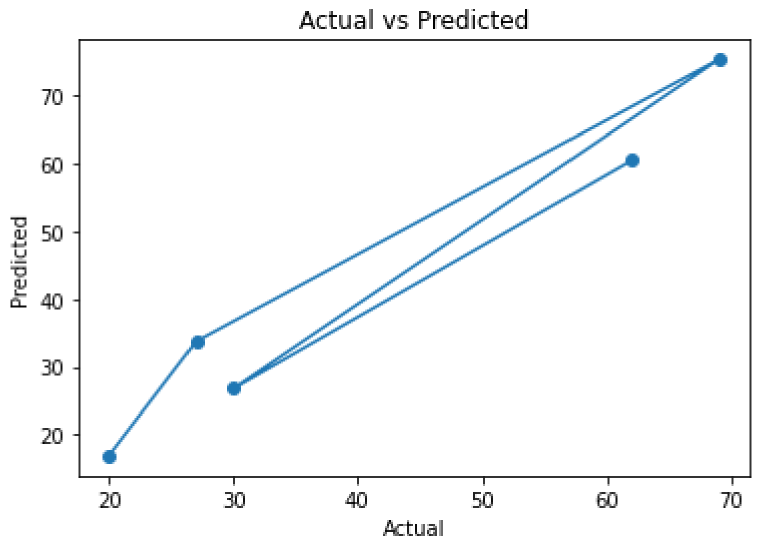
```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_prediction})
data
```

Out[22]:

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

In [23]:

```
plt.scatter(y_test,y_prediction)
plt.plot(y_test,y_prediction)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted")
plt.show()
```



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

In [29]:

```
hours = 9.25
predicted = LRM.coef_*hours+LRM.intercept_
print("No of Hours = ",hours)
print("Predicted Score = ",predicted[0])
y_predict=LRM.predict([[hours]])
print("Predicted Score if a student studied for 9.25 hours per day is :",y_predict[0])
```

```
No of Hours =  9.25
Predicted Score =  93.69173248737538
Predicted Score if a student studied for 9.25 hours per day is : 93.69173248
737538
```

Model Evaluation

The final step is to evaluate the performance of algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For simplicity here, we have chosen the mean square error. There are many such metrics.

In [25]:

```
from sklearn import metrics
print('Mean Absolute Error:',
      metrics.mean_absolute_error(y_test, y_prediction))
```

```
Mean Absolute Error: 4.183859899002975
```

In [ ]: