The Spark Foundation-GRIP- Data Science and Business Analytics-Aug21

Task by: Deepti Taram

Language used: Python 3

IDE: Jupyter Notebook

TYPE: Linear Regression

Task 1: Prediction using Supervised Machine Learning

- we have to predict the percentage of the score of a student based on number of hours he/she studied. This task has two variable where the target value is the percentage score and feature is the number of hours studied.

Step 1:Import the Dataset

In [1]:
```
# Firstly Importing all libraries required in this notebook
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# Reading data from given link
url = "http://bit.ly/w-data"
student_data = pd.read_csv(url)
# Lets observe the dataset
student_data.head(25)

print("Data imported successfully")
```

```
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

Step 2: Explore the data

In [3]:
```
student_data.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 [4]:
```
student_data.describe()
```

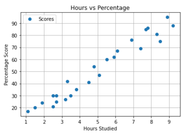
Out[4]:

	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

Step 3:Visualize the data

Let's plot our data points on 2-D graph to eyeball our dataset and see if we can manually find any relationship between the data. We can create the plot with the following script:

In [6]:
```
# Plotting the distribution of scores
student_data.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.grid()
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.

Step 4:Preparing the data

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

In [7]:
```
X = student_data.iloc[:, :-1].values
y = student_data.iloc[:, 1].values
```

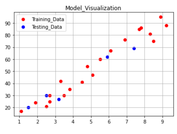
In [8]:
```
X
```

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]])
```

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

In [9]:
```
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)
```

In [10]:
```
plt.scatter(X_train,y_train,label = 'Training_Data', color='r')
plt.scatter(X_test,y_test,label = 'Testing_Data',color ='b')
plt.legend()
plt.title("Model_Visualization")
plt.grid()
plt.show()
```



Step 5: Training the Algorithm¶

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

In [11]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

print("Training complete")
```

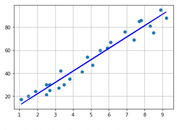
```
Training complete
```

Now our model is trained, it's time to visualize the best-fit line of regression

Step 6:Visualize the Model

In [12]:
```
# plotting regression line
line = regressor.coef_*X+regressor.intercept_

# Plotting for the test data
plt.scatter(X, y,label = 'data')
plt.plot(X, line,label='line',color='b');
plt.grid()
plt.show()
```



Step 7: Making Predictions¶

Now that we have trained our algorithm, it's time to make some predictions

In [13]:
```
y_pred = regressor.predict(X_test)
print(X_test)
```

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

In [14]:
```
df = pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
df
```

Out[14]:

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

In [15]:
```
# Estimating the traning and test score
print("Training score:",regressor.score(X_train,y_train))
print("Test score:",regressor.score(X_test,y_test))
```

```
Training score: 0.9515510725211552
Test score: 0.9454906892105356
```

In [16]:
```
# Testing the model with our own data
hours = 9.25
test = np.array([hours])
test = test.reshape(-1,1)
own_pred = regressor.predict(test)
print("No of Hours = {}".format(hours))
print("Predicted Score = {}".format(own_pred[0]))
```

```
No of Hours = 9.25
Predicted Score = 93.69173248737538
```

Step 8: Evaluating the Algorithm

- 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 [17]:
```
from sklearn import metrics
print('Mean Absolute Error:',
      metrics.mean_absolute_error(y_test, y_pred))
```

```
Mean Absolute Error: 4.183859899002975
```

Result:

- After analysing we have got predicted score around 93 based on the number of study hours i.e., 9.25 hrs/day