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.

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 simple linear regression task as it involves just two variables.

Author: Arathi Unnikrishnan

Student Marks Prediction

Task -1

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

Importing Data

In [8]:
```
url="http://bit.ly/w-data"
data=pd.read_csv(url)
print("Data imported successfully")
data.head(10)
```

```
Data imported successfully
```

Out[8]:

	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

In [9]:
```
data
```

Out[9]:

	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

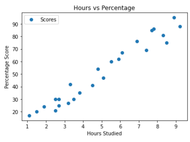
In [10]:
```
data.describe()
```

Out[10]:

	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

Visualizing the Data

In [13]:
```
data.plot(x='Hours',y='Scores',style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



Training and Testing

In [14]:
```
x = data.iloc[:,:-1].values
y = data.iloc[:,1].values
```

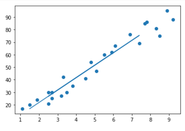
In [15]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=0)
```

Modeling

In [17]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train,y_train)
Y_pred = regressor.predict(x_test)
```

Plotting Regression Graph

In [18]:
```
plt.scatter(x,y)
plt.plot(x_test,Y_pred);
plt.show()
```



In [19]:
```
print(x_test) # Testing data - In Hours
y_pred = regressor.predict(x_test) # Predicting the scores
```

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

Making Prediction

In [22]:
```
# Comparing Actual vs Predicted
df = pd.DataFrame({'Actual': y_test,'Predicted':y_pred})
df
```

Out[22]:

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

Perdicting the marks of the student, if a student studies for 9.25hrs in a day

In [24]:
```
predicted =[[9.25]]
result = regressor.predict(predicted)
result
```

Out[24]:
```
array([93.69173249])
```

Evaluating the model

In [26]:
```
from sklearn import metrics
Linear = metrics.mean_absolute_error(y_test,Y_pred)
print('Mean absolute Error:',Linear)
```

```
Mean absolute Error: 4.183859899002975
```

In [27]:
```
from sklearn import metrics
print('Mean absolute Error:',
      metrics.mean_absolute_error(y_test,y_pred))
```

```
Mean absolute Error: 4.183859899002975
```

Random Forest Regressor

In [28]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [29]:
```
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
regressor.fit(x_train,y_train)
Y_pred= regressor.predict(x_test)
```

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

Out[30]:

	Actual	Predicted
0	20	19.210000
1	27	37.580000
2	69	84.260000
3	30	23.435833
4	62	63.120000

Mean Absolute Error

In [31]:
```
from sklearn import metrics
Random_Forest=metrics.mean_absolute_error(y_test,Y_pred)
print('Mean Absolute Error:',Random_Forest)
```

```
Mean Absolute Error: 6.862833333333333
```

Summarized Table

In [32]:
```
df1=pd.DataFrame({'Model':['Linear','Random_Forest'],'Mean_Absolute_Error':['4.18','6.86']})
df1
```

Out[32]:

	Model	Mean_Absolute_Error
0	Linear	4.18
1	Random_Forest	6.86