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 a simple linear regression task as it involves just two variables.

In [122…
```
# Importing all libraries required in this notebook
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

In [123…
```
# Reading data from remote link
url = "http://bit.ly/w-data"
s_data = pd.read_csv(url)
print("Data imported successfully")

s_data.head(10)
```

```
Data imported successfully
```

Out[123…

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

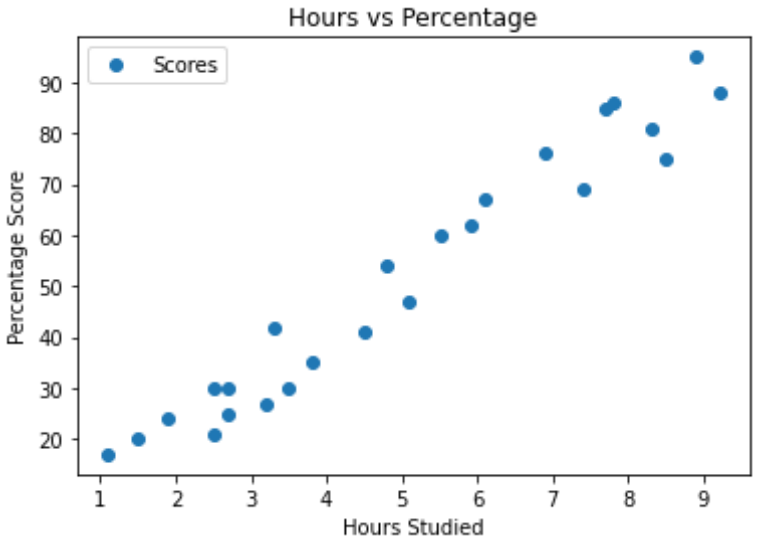
	Hours	Scores
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25

Let's plot our data points on 2-D graph. We can create the plot with the following script:

In [126…
```
#Returning the dimension of array in the form of(n,m)
s_data.shape
```

Out[126…
```
(25, 2)
```

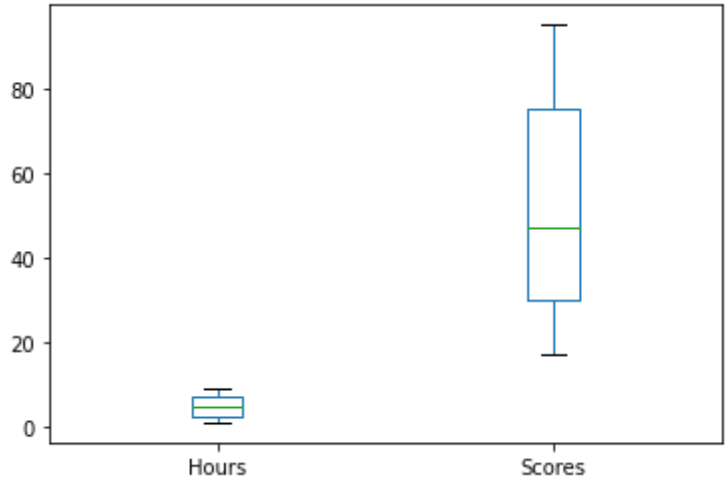
In [130…
```
#Plotting the destribution of scores
s_data.plot(style='o',x='Hours',y='Scores')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



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.

In [132…

```
#Plotting the destribution of scores with different style
s_data.plot(kind='box')
plt.show()
```



Preparing the data

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

In [133…
```
# split into inputs and outputs
X = s_data.iloc[:, :-1].values
y = s_data.iloc[:, 1].values
print(X.shape, y.shape)
```

```
(25, 1) (25,)
```

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 train_test_split() method:

In [135…
```
# split into train and test sets
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 data

We have split our data and its time to train our our algorithm.

In [101…
```
#Change to dataframe variables
Hs=pd.DataFrame(s_data['Hours'])
Ss=pd.DataFrame(s_data['Scores'])
```

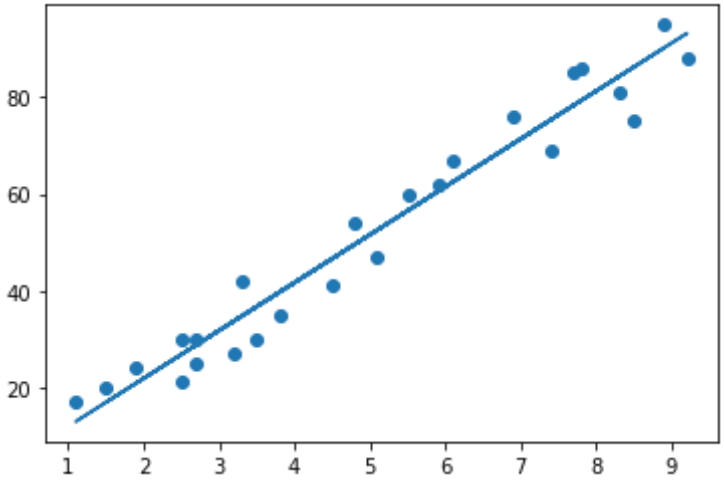
In [114…
```
#Build linear regression model
lm=linear_model.LinearRegression()
model=lm.fit(Hs,Ss)
print('Training is complete')
```

```
Training is complete
```

In [136…
```
# Plotting the regression line
line = regressor.coef_*X+regressor.intercept_

# Plotting for the test data
```

```
plt.scatter(Hs, Ss)
plt.plot(Hs, line);
plt.show()
```



Making Predictions

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

In [104…
```
print(X_test) # Testing data - Hours
y_pred = regressor.predict(X_test) # Predicting the scores
```

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

In [105…
```
# Comparing Actual vs Predicted
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[105…

	Actual	Predicted
0	20	16.884145

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

In [80]:
```
#Returning pairwise correlation of all columns
s_data.corr()
```

Out[80]:

	Hours	Scores
Hours	1.000000	0.976191
Scores	0.976191	1.000000

In [83]:
```
#Returns estimated coefficients
c1=model.coef_
print(c1)
c2=model.intercept_
print(c2)
```

```
[[9.77580339]]
[2.48367341]
```

In [137…
```
#Measuring accuracy of the model against the training data
model.score(Hs,Ss)
```

Out[137… 0.9529481969048356

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

In [138…
```
#Testing with own data
own_pred = model.predict(np.array(9.25).reshape(-1,1))
print("No of Hours = {}".format(hours))
print("Predicted Score = {}".format(own_pred))
```

```
No of Hours = 9.25
Predicted Score = [[92.90985477]]
```

Evaluating the model

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 [87]:
```
from sklearn import metrics
print('Mean Absolute Error:',
      metrics.mean_absolute_error(y_test, y_pred))
```

```
Mean Absolute Error: 4.183859899002975
```