GRIP @ The Sparks Foundation.

Task1 - Predict the percentage of a student based on the no. of study hours using Supervised Machine Learning.

PASNOOTI DEVENDER

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.

Importing Dataset

In [127...
```
######### https://raw.githubusercontent.com/AdiPersonalWorks/Random/master/student_scores%20-%20student_scores.csv
```

In [106...
```
# 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 [146...
```
# data extraction of url
score_data = pd.read_csv(r"http://bit.ly/w-data")
score_data.head()
```

Out[146...

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

In [109...
```
score_data.tail()
```

Out[109...

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

In [110...
```
score_data.describe()
```

Out[110...

	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 [111...
```
score_data.shape
```

Out[111...
```
(25, 2)
```

In [112...
```
score_data.info()
```

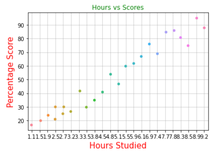
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 528.0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

In [113...
```
score_data.isnull().sum()
```

Out[113...
```
Hours     0
Scores    0
dtype: int64
```

Visualization of Data set

In [145...
```
import seaborn as sns
data1 = score_data
ax = sns.stripplot(x='Hours', y='Scores', data= data1)
plt.title("Hours vs Scores", color = "g")
font1 = {'color':'red','size':15}
font2 = {'color':'red','size':15}
plt.xlabel('Hours Studied',fontdict = font1 )
plt.ylabel('Percentage Score', fontdict = font2 )
plt.grid(linewidth=0.5)
```



Preparing the data

In [115...
```
score_data.corr()
```

Out[115...

	Hours	Scores
Hours	1.000000	0.976191
Scores	0.976191	1.000000

In [116...
```
X = score_data.iloc[:, :-1].values
X
```

Out[116...
```
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 [117...
```
y = score_data.iloc[:, 1].values
y
```

Out[117...
```
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 [118...
```
# splitting the data into training 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

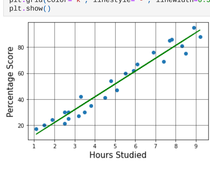
We have split our data into training and testing sets, and now is finally the time to train our algorithm.

In [119...
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_train, y_train)
print("Training complete.")
```

```
Training complete.
```

In [126...
```
# Plotting the regression line
line = regressor.coef_*X+regressor.intercept_

# Plotting for the test data
plt.scatter(X, y)
plt.plot(X, line, color = "g")
font1 = {'color':'k','size':15}
font2 = {'color':'k','size':15}
plt.xlabel('Hours Studied',fontdict = font1 )
plt.ylabel('Percentage Score', fontdict = font2 )
plt.grid(color='k', linestyle='-', linewidth=0.5)
plt.show()
```



Making Predictions

In [121...
```
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]]
```

In [122...
```
# Comparing Actual vs Predicted
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[122...

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

In [123...
```
# You can also test with your own data
hours = 9.25
own_pred = regressor.predict([[hours]])
print("No of Hours = {}".format(hours))
print("Predicted Score = {}".format(own_pred[0]))
```

```
No of Hours = 9.25
Predicted Score = 93.69173248737539
```

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 [124...
```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

```
Mean Absolute Error: 4.183859899002982
```

THE END