GRIP: The Sparks Foundation

$(Data Science and Business Analytics Intern)$

Author:- Koye Harshitha

Task 1:- Prediction using Supervised Machine Learning

- **This task is about to predict the percentage score of the student based on the number hours studied. This task involves 2 variables and it can be done by using Simple Linear Regression**

In [1]:

```
# importing all the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:

```
#Reading Data
url="https://raw.githubusercontent.com/AdiPersonalWorks/Random/master/student_scores%20-%20
DF=pd.read_csv(url)
print("Data Imported sucessfully")
DF
```

```
Data Imported sucessfully
```

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

Exploring Data

In [3]:

```
# This will return the number of rows and columns of the given dataset
DF.shape
```

Out[3]:

(25, 2)

In [4]:

```
# The head() function is used to get the first 5 rows from the data set
DF.head()
```

Out[4]:

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

In [5]:

```
# The tail() function is used to get the bottom 5 rows from the data set
DF.tail()
```

Out[5]:

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

In [6]:

```
# The describe() method is used for calculating some statistical data like percentile, mean
DF.describe()
```

Out[6]:

	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 [7]:

```
# this function is used to get a concise summary of the dataframe
DF.info()
```

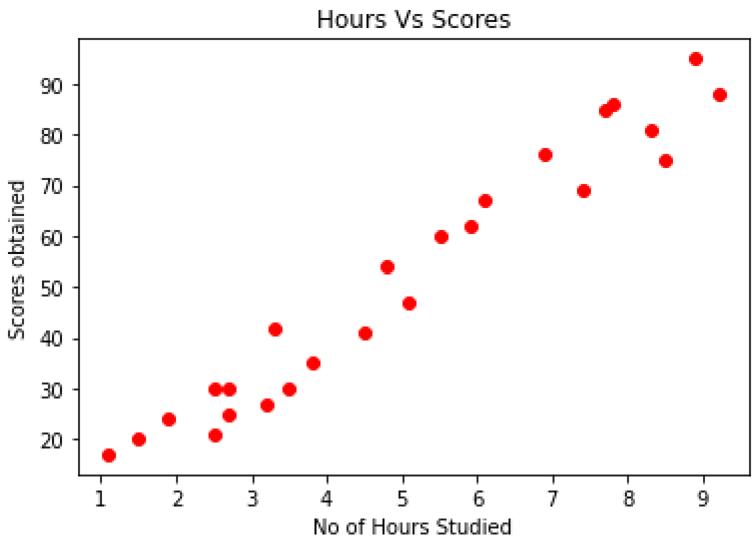
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 528.0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

Visualizing the Data

- **plotting the data points to find the relation between the two variables**

In [8]:

```
plt.scatter(x="Hours", y="Scores", color="red", data=DF)
plt.title("Hours Vs Scores")
plt.xlabel("No of Hours Studied")
plt.ylabel("Scores obtained")
plt.show()
```



From the above graph we can clearly see that there is linear relationship between the two variables

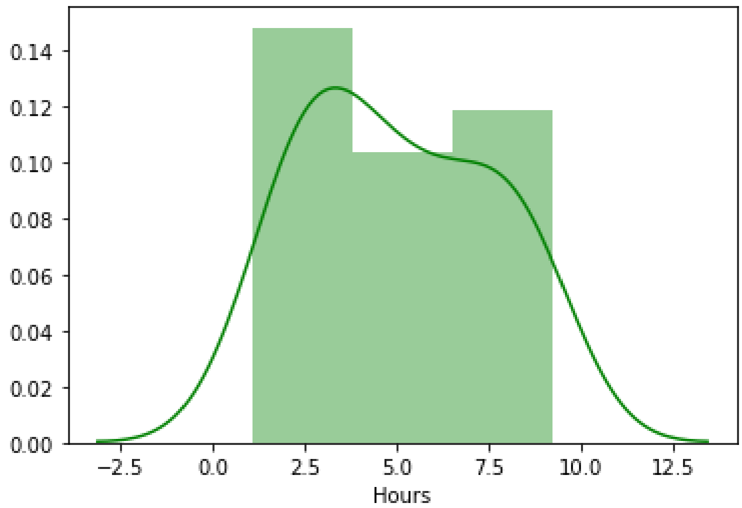
- ## Distribution plot for the two variables to check if there are any outliers or not

In [9]:

```
sns.distplot(DF['Hours'], color='green')
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x22d42650a60>
```

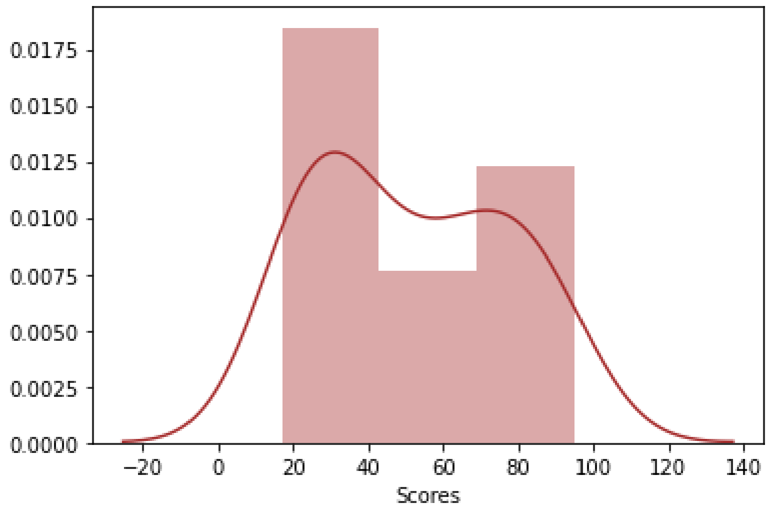


In [10]:

```
sns.distplot(DF['Scores'], color='brown')
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x22d426c45e0>
```



Preparing the data using Simple Linear Regression

In [11]:

```
# First we need to divide our data into independent(x) and dependent(y) variables
x=DF.iloc[:,0:1].values
y=DF.iloc[:, 1].values
```

In [12]:

```
# 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
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)
#here test size 0.2 means only 20% of data is going to be tested
```

Training the Algorithm

In [13]:

```
# we have split our data into training and testing sets, and now its time to train our data
from sklearn.linear_model import LinearRegression
Reg = LinearRegression()
Reg.fit(X_train, y_train)
```

Out[13]:

```
LinearRegression()
```

In [14]:

```
print("Training completed.")
```

```
Training completed.
```

In [15]:

```
Coefficient=Reg.coef_
Coefficient
```

Out[15]:

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

In [16]:

```
Intercept=Reg.intercept_
Intercept
```

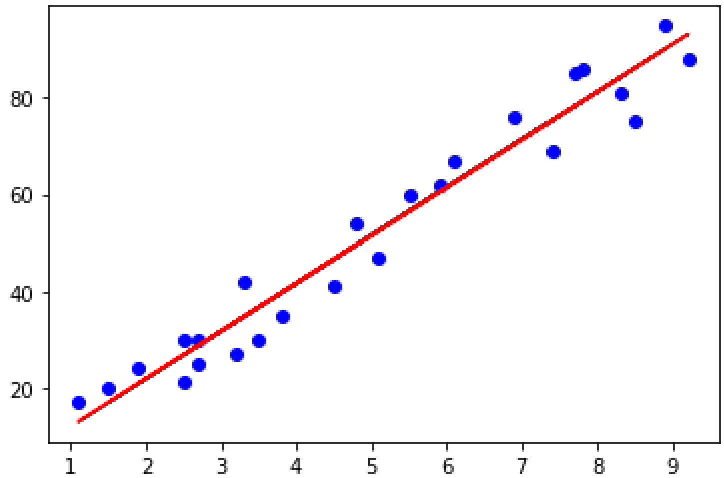
Out[16]:

```
2.018160041434683
```

In [17]:

```
# This is the linear regression line equation
Line=Coefficient*x+Intercept

#Plotting the regression line
plt.scatter(x,y, color='blue')
plt.plot(x, Line, color='red');
plt.show()
```



Making Predictions

In [18]:

```
# Testing Data in hours
print(X_test)
y_pred=Reg.predict(X_test)
print(y_pred)
```

```
[[1.5]
 [3.2]
 [7.4]
 [2.5]
 [5.9]]
[16.88414476 33.73226078 75.357018   26.79480124 60.49103328]
```

In [19]:

```
# Comparing the values of Actual(y_test) vs Predicted(y_pred)
comparison_dataset=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
comparison_dataset
```

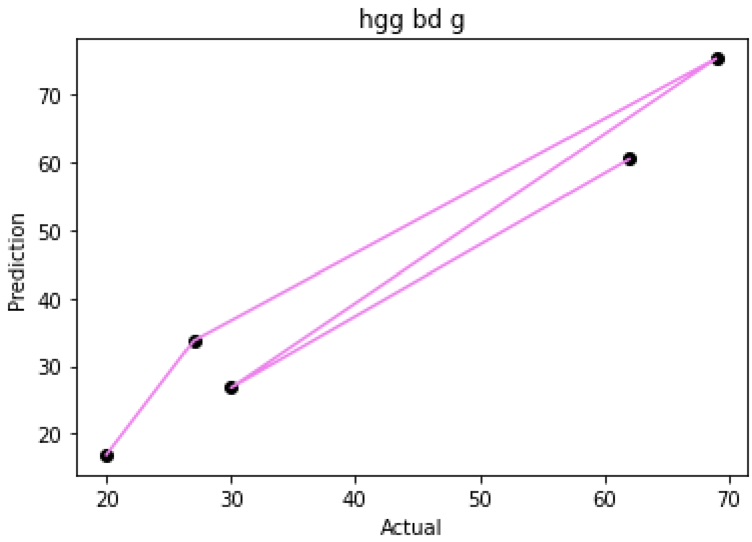
Out[19]:

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

Plotting the Graph Between Actual(y_test) vs Predicted(y_pred) values

In [20]:

```
plt.scatter(y_test,y_pred, color='black')
plt.plot(y_test,y_pred, color='violet')
plt.xlabel("Actual")
plt.ylabel("Prediction")
plt.title("hgg bd g")
plt.show()
```



you can also test the data with your own values also

In [21]:

```
hours=9.25
OwnData_pred=Reg.predict([[hours]])
print("No of Hours = {}".format(hours))
print("Predicted Score = {}".format(OwnData_pred[0]))
```

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

Model Evalution

The final step is to evaluate the performance of the algorithm. The step is particularly important to compare how well different algorithms perform on a particular dataset

In [22]:

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

```
Mean Absolute Error: 4.183859899002975
```

In [ ]:

```

```