GRIP : The Spark Foundation

Data Science & Business Analytics Internship

Graduate Rotational Internship Program

Name: Gaikwad Pawan Ramesh

Task 1: Prediction using Supervised ML

Import Dataset : Numpy, Pandas, Matplotlib, Seaborn, scikit learn.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings as wg
wg.filterwarnings('ignore')
```

Read Dataset from URL

```
url= 'http://bit.ly/w-data'
data = pd.read_csv(url)
print("Successfully Import Dataset")
data
```

```
Successfully Import Dataset
```

	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

print First 5 record in dataset

```
data.head()
```

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

print last 5 record in dataset

```
data.tail()
```

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

Use describe() method we can see that percentiles,mean,std,max,count of given dataset

```
data.describe()
```

	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

Full summary of our dataframe

```
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
```

used to get a Series containing counts of unique values

```
data['Hours'].value_counts()
```

```
2.5    2
2.7    2
3.8    1
8.5    1
3.5    1
1.5    1
5.5    1
4.5    1
5.9    1
6.1    1
7.7    1
1.9    1
1.1    1
4.8    1
8.3    1
8.9    1
7.4    1
7.8    1
5.1    1
3.2    1
9.2    1
3.3    1
6.9    1
Name: Hours, dtype: int64
```

```
data['Scores'].value_counts()
```

```
30    3
95    1
62    1
85    1
86    1
67    1
24    1
69    1
17    1
41    1
42    1
75    1
47    1
76    1
81    1
20    1
21    1
54    1
88    1
25    1
27    1
60    1
```

```
35    1
Name: Scores, dtype: int64
```

Used median() method we can see median in our dataset

```
data.median()
```

```
Hours      4.8
Scores    47.0
dtype: float64
```

max() method find maximum value in our dataframe

```
data.max()
```

```
Hours      9.2
Scores    95.0
dtype: float64
```

min() method find minimumvalue in our dataframe

```
data.min()
```

```
Hours      1.1
Scores    17.0
dtype: float64
```

find shape of dataset

```
data.shape
```

```
(25, 2)
```

Checking the missing values

```
data.isnull().sum()
```

```
Hours     0
Scores    0
dtype: int64
```

find the correlation

```
data.corr()
```

	Hours	Scores
Hours	1.000000	0.976191
Scores	0.976191	1.000000

how many columns in our dataset

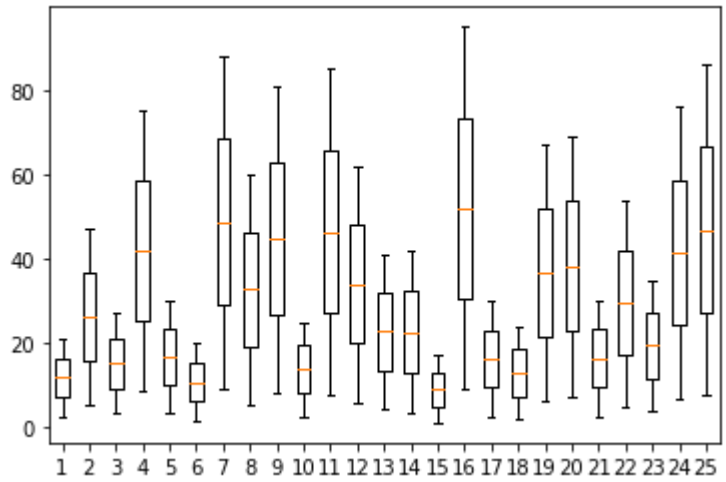
```
data.columns
```

```
Index(['Hours', 'Scores'], dtype='object')
```

Visualize Data

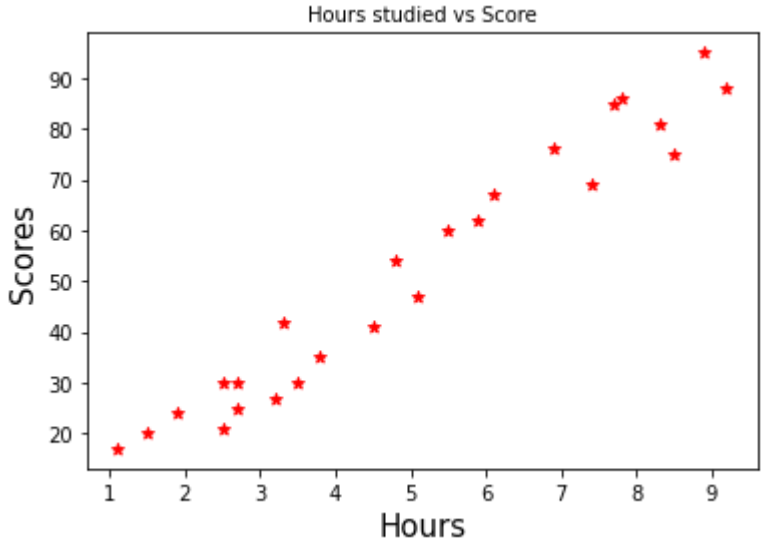
Perform box plot graph using Seaborn Libraries

```
plt.boxplot(data)
plt.show()
```



perform a scatter plot graph

```
plt.xlabel('Hours',fontsize=15)
plt.ylabel('Scores',fontsize=15)
plt.title('Hours studied vs Score',fontsize=10)
plt.scatter(data.Hours,data.Scores,color="red",marker="*")
plt.show()
```



"Scatter plot" indicate linear relationship as hours, your changes is high scoring

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

```
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]])
```

```
y
```

```
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])
```

Preparing Data and splitting into train and test sets

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


## we have splitting out data using 80:20 RULE
print('x train.shape',x_train.shape)
print('x test.shape',x_test.shape)
print('y train.shape',y_train.shape)
print('y test.shape',y_test.shape)
```

```
x train.shape (5, 1)
x test.shape (20, 1)
y train.shape (5,)
y test.shape (20,)
```

Training Model

```
from sklearn.linear_model import LinearRegression
linreg=LinearRegression()


## fit training data
linreg.fit(x_train,y_train)
print('Training our algorithm is end')
```

```
Training our algorithm is end
```

```
## A0 is intercept and A1 is slope of line
print('A0 =',linreg.intercept_,'\nA1 =',linreg.coef_)
```

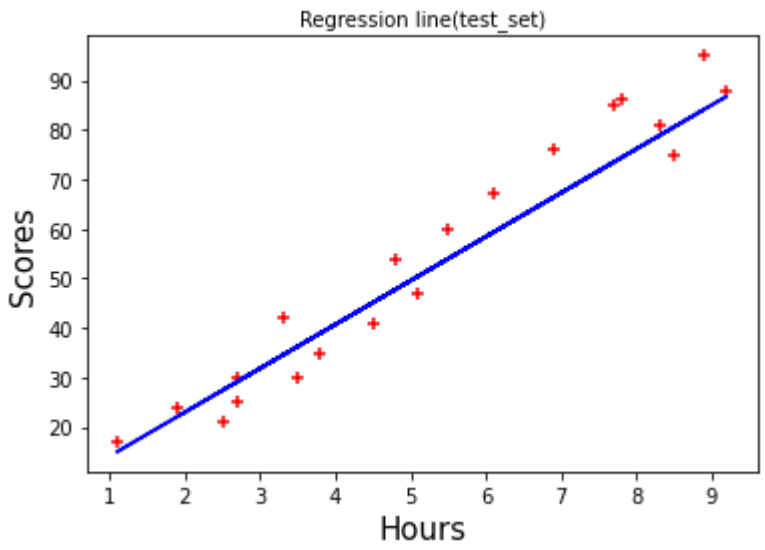
```
A0 = 5.264468260511144
A1 = [8.86232481]
```

Plotting the line of regression

```
Y0 = linreg.intercept_+linreg.coef_*X_train
```

```
Y0 = linreg.intercept_ + linreg.coef_ * x_train
```

```
##test data
plt.plot(x_test,y_pred,color='blue')
plt.scatter(x_test,y_test,color="red",marker="+")
plt.xlabel('Hours',fontsize=15)
plt.ylabel('Scores',fontsize=15)
plt.title('Regression line(test_set)',fontsize=10)
plt.show()
```



Test Data

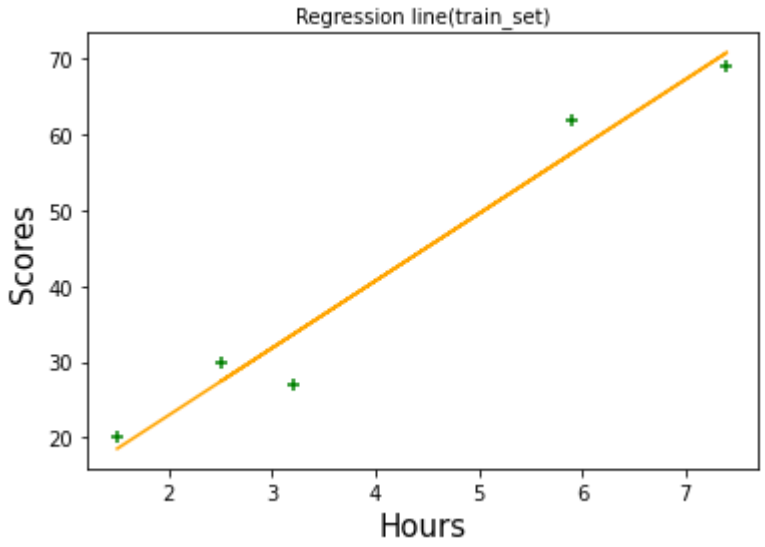
```
## predict score of data
y_pred =linreg.predict(x_train)
print(y_pred)
```

```
[18.55795548 33.62390767 70.84567189 27.4202803  57.55218467]
```

```
y_train
```

```
array([20, 27, 69, 30, 62])
```

```
## test data
plt.scatter(x_train,y_train,color="green",marker="+")
plt.plot(x_train,Y0,color='orange')
plt.xlabel('Hours',fontsize=15)
plt.ylabel('Scores',fontsize=15)
plt.title('Regression line(train_set)',fontsize=10)
plt.show()
```



Comparing Actual Score and Predict Scores

```
y_test1=list(y_train)
prediction=list(y_pred)
df_compare = pd.DataFrame({'Actual':y_test1, 'Result':prediction})
df_compare
```

	Actual	Result
0	20	18.557955
1	27	33.623908
2	69	70.845672
3	30	27.420280
4	62	57.552185

Accuracy of Model

```
from sklearn import metrics
metrics.r2_score(y_train,y_pred)
```

```
0.9617402761556321
```

Predict score

```
predict_score = linreg.predict([[9.25]])
```

```
print('predict the score for student 9.25 hours',predict_score)
```

```
predict the score for student 9.25 hours [87.24097279]
```

Thank you