Name:: SHIVAM GAIKWAD

GMAIL:: shivamgaikwad39@gmail.com

Task 1:: Predict the percentage of an student based on the no. of study hours.

Linear Regression

Simple Linear Regression

Description: A machine learning model to predict the percentage of marks that a student is expected to score based upon the number of hours they studied.

Importing the required libraries & dataset

In [1]:
```
# Importing required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib.inline

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

In [2]:
```
# Importing dataset
dataset = pd.read_csv('http://bit.ly/w-data')
dataset.head(10)
```

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

Data Analysis

In [3]:
```
# Shape of the dataset
dataset.shape
```

Out[3]:
```
(25, 2)
```

In [4]:
```
# Descriptive statistic summary
dataset.describe()
```

Out[4]:

	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 [5]:
```
# Checking the null values, data-types etc. of the column i.e concise summary of the columns
dataset.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 [6]:
```
dataset.isnull().sum()
```

Out[6]:
```
Hours     0
Scores    0
dtype: int64
```

Data Manipulation

In [7]:
```
x = dataset.iloc[:,:1]
y = dataset.iloc[:,1:]
```

In [8]:
```
x.head()
```

Out[8]:

	Hours
0	2.5
1	5.1
2	3.2
3	8.5
4	3.5

In [9]:
```
y.head()
```

Out[9]:

	Scores
0	21
1	47
2	27
3	75
4	30

In [10]:
```
# Sorting the dataset -
# Sorting 'Scores' column in descending order
sort = dataset.sort_values(by = 'Scores', ascending = False)
sort.head(11)
```

Out[10]:

	Hours	Scores
15	8.9	95
5	9.2	88
24	7.8	86
10	7.7	85
8	8.3	81
20	6.9	76
3	8.5	75
18	7.4	69
16	6.1	67
11	5.9	62

In [11]:
```
# Filtering the dataset -
# Filtering all those records from the dataset whose score is greater than 80 and hours is gr
# than 8 hours
filtered_data = (dataset['Scores'] > 80) & (dataset['Hours'] > 8)
# Applying filter to the dataset
filter_df = dataset[filtered_data]
```

Out[11]:

	Hours	Scores
5	9.2	88
8	8.3	81
15	8.9	95

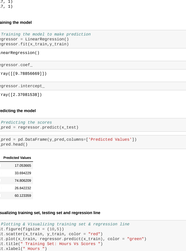
In [12]:
```
# Sampling random records
sample = dataset.sample(n=10)
sample_10
```

Out[12]:

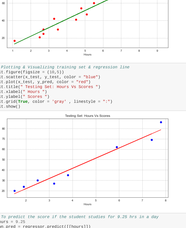
	Hours	Scores
14	1.1	17
10	7.7	85
3	8.5	75
12	4.5	41
15	8.9	95
1	5.1	47
11	5.9	62
9	2.7	25
0	2.5	21

Data Visualization

In [13]:
```
# Scatter Plot
plt.figure(figsize = (15,5),facecolor = "#CFEAEA",linewidth = 10.0,tight_layout = False)
plt.scatter(x.y, c = 'RIMSON', marker = '*', s = 100, alpha = 0.5)
plt.title(' Hours Vs Scores ')
plt.xlabel(' Hours Studied ')
plt.ylabel(' Scores ')
plt.legend('Scores'), loc = 'best')
plt.grid(True, color = 'gray', linestyle = "-.")
plt.show()
```



In [14]:
```
# Violin Plot
plt.figure(figsize = (15,5))
hrs = dataset['Hours'].to_list()
scores = dataset['Scores'].to_list()
plt.violinplot(hrs)
plt.violinplot(scores, showmeans = True, showmedians = True)
plt.title('Violin Plot')
plt.show()
plt.show()
```



In [15]:
```
# Regression Plot
plt.figure(figsize = (15,5))
sns.set_style('white')
sns.regplot(x="Hours", y="Scores", data=dataset, color='#FB5B62' )
```

Out[15]:
```
<AxesSubplot:xlabel='Hours', ylabel='Scores'>
```

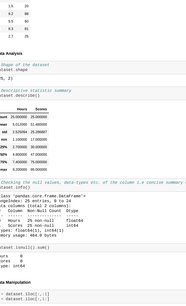
In [16]:
```
# Correlation
corr_df = dataset.corr()
corr_df
```

Out[16]:

	Hours	Scores
Hours	1.000000	0.976191
Scores	0.976191	1.000000

In [17]:
```
# Heat Map
plt.figure(figsize=(15,5))
sns.heatmap(corr_df, cmap='Reds',annot=True)
```

Out[17]:
```
<AxesSubplot:>
```



Preparing the data

In [18]:
```
x = dataset.iloc[:,:1]
y = dataset.iloc[:,1:]
```

In [19]:
```
x.head()
```

Out[19]:

	Hours
0	2.5
1	5.1
2	3.2
3	8.5
4	3.5

In [20]:
```
y.head()
```

Out[20]:

	Scores
0	21
1	47
2	27
3	75
4	30

In [21]:
```
# Splitting data into training & test sets
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.30 ,random_state=0)
```

In [22]:
```
print("Test Set")
print(x_test.shape)
print(y_test.shape)

Test Set
(8, 1)
(8, 1)
```

In [23]:
```
print("Training Set")
print(x_train.shape)
print(y_train.shape)

Training Set
(17, 1)
(17, 1)
```

Training the model

In [24]:
```
# Training the model to make prediction
regressor = LinearRegression()
regressor.fit(x_train,y_train)
```

Out[24]:
```
LinearRegression()
```

In [25]:
```
regressor.coef_
```

Out[25]:
```
array([[9.78554669]])
```

In [26]:
```
regressor.intercept_
```

Out[26]:
```
array([2.37981538])
```

Predicting the model

In [27]:
```
# Predicting the scores
y_pred = regressor.predict(x_test)
```

In [28]:
```
y_pred = pd.DataFrame(y_pred,columns=['Predicted Values'])
y_pred.head()
```

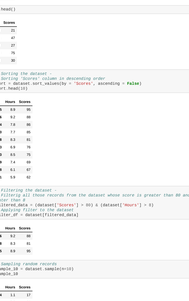
Out[28]:

	Predicted Values
0	17.053665
1	33.694229
2	74.806209
3	26.842232
4	60.123359

Visualizing training set, testing set and regression line

In [29]:
```
# Plotting & Visualizing training set & regression line
plt.figure(figsize = (15,5))
plt.scatter(x_train, y_train, color = "blue")
plt.plot(x_train, regressor.predict(x_train), color = "green")
plt.title(' Training Set: Hours Vs Scores ')
plt.xlabel(' Hours ')
plt.ylabel(' Scores ')
plt.grid(True, color = 'gray' , linestyle = "-")
plt.show()
```


In [30]:
```
# Plotting & Visualizing training set & regression line
plt.figure(figsize = (15,5))
plt.scatter(x_test, y_test, color = "blue")
plt.plot(x_test, y_pred, color = "red")
plt.title(' Testing Set: Hours Vs Scores ')
plt.xlabel(' Hours ')
plt.ylabel(' Scores ')
plt.grid(True, color = 'gray' , linestyle = "-")
plt.show()
```



In [31]:
```
# To predict the score if the student studies for 8.25 hrs in a day
hours = 8.25
pred = regressor.predict([[hours]])
print("No of Hours = {}".format(hours))
print("Predicted Score = {}".format(own_pred[0]))

No of Hours = 8.25
Predicted Score = [82.91595723]
```

Evaluating the model

In [32]:
```
print("Mean Absolute Error: ", metrics.mean_absolute_error(y_test,y_pred))

Mean Absolute Error:  4.419727808027654
```

In [33]:
```
error = metrics.mean_squared_error(y_test,y_pred)
print ("Mean Square Error:",error)

Mean Square Error:  22.96508721729417
```

In [34]:
```
print ('Root Mean Square Error: ', np.sqrt(metrics.mean_squared_error(y_test,y_pred)))

Root Mean Square Error:  4.792191274636314
```

In [35]:
```
print("R^2 score:",metrics.r2_score(y_test, y_pred))

R^2 score:  0.9568211104435258
```