The Sparks Foundation

Data Science and Business Analytics

Done By : Neeraj Boyapati

Task 1 - Prediction using supervised ML

- Predict the percentage of the student based on the no of study hours
- This task involves 2 variables which can be done by using Simple Linear Regression
- Prediction of the percentage of student who studies 9.25 hrs/day ?

Importing Required Libraries

In [5]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Reading the Dataset

In [11]:
```
#Reading Data
url="https://raw.githubusercontent.com/AdiPersonalWorks/Random/master/student_sco
DF=pd.read_csv(url)
print("Data Imported sucessfully")
DF
```

```
Data Imported sucessfully
```

Out[11]:

	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 the Data

In [21]:
```
# This will return the number of rows and columns of the given dataset.
DF.shape
```

Out[21]: (25, 2)

In [22]:
```
# The head() function is used to get the first 5 rows from the data set.
DF.head()
```

Out[22]:

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

In [23]:
```
# The tail() function is used to get the bottom 5 rows from the data set.
DF.tail()
```

Out[23]:

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

Data Description

In [24]:
```
# The describe() method is used for calculating some statistical data like percen
DF.describe()
```

Out[24]:

	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 [25]:
```
# 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):
Hours     25 non-null float64
Scores    25 non-null int64
dtypes: float64(1), int64(1)
memory usage: 480.0 bytes
```

In [26]:
```
# the dtypes attribute returns a Series with the data type of each column.
DF.dtypes
```

Out[26]:
```
Hours     float64
Scores      int64
dtype: object
```

In [27]:
```
# nunique() function return number of unique elements in the object.
DF.nunique()
```

Out[27]:
```
Hours     23
Scores    23
dtype: int64
```

Deleting Missing Values

In [31]:
```
# The dropna() function is used to remove missing values.
DF.dropna()
```

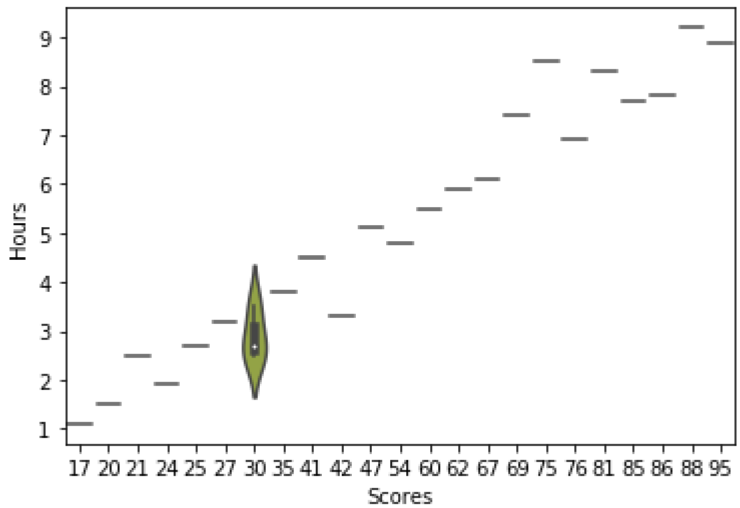
Out[31]:

	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

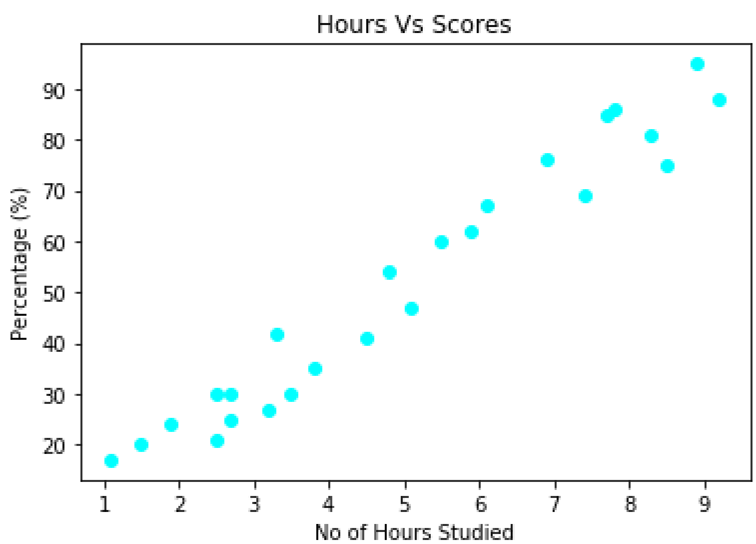
Data Visualisation

In [45]:
```
# Violin Plot between Scores and Hours Studied.
sns.violinplot(x='Scores',y='Hours',data=DF)
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0xd04b5002b0>

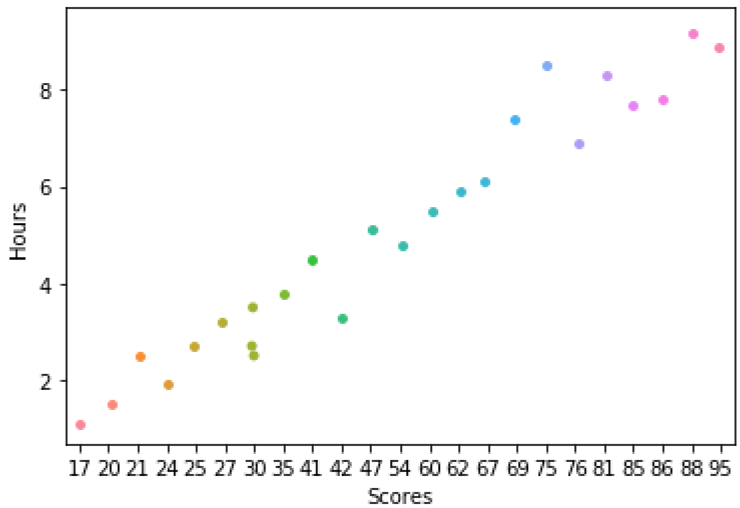


In [46]:
```
# Scatter plot between Percentage(%) and Hours Studied.
plt.scatter(x="Hours", y="Scores", color="cyan", data=DF)
plt.title("Hours Vs Scores")
plt.xlabel("No of Hours Studied")
plt.ylabel("Percentage (%)")
plt.show()
```



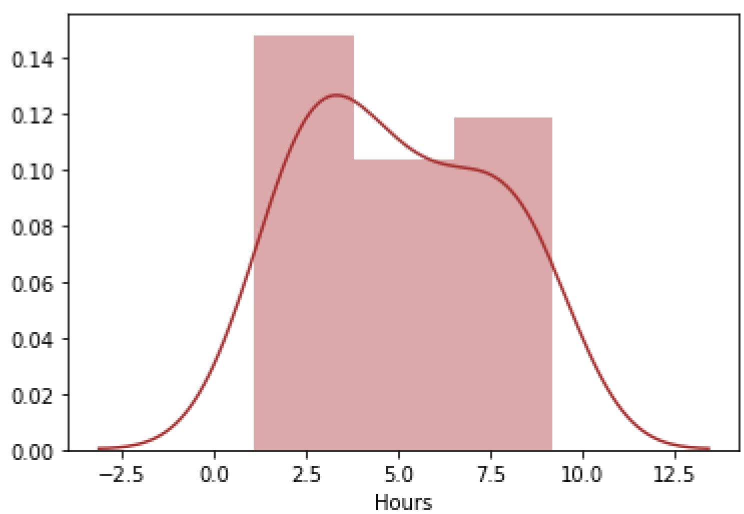
In [47]:
```
# Strip Plot between Scores and Hours Studied.
sns.stripplot(x='Scores',y='Hours',data=DF,jitter=True)
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0xd04b6fc048>



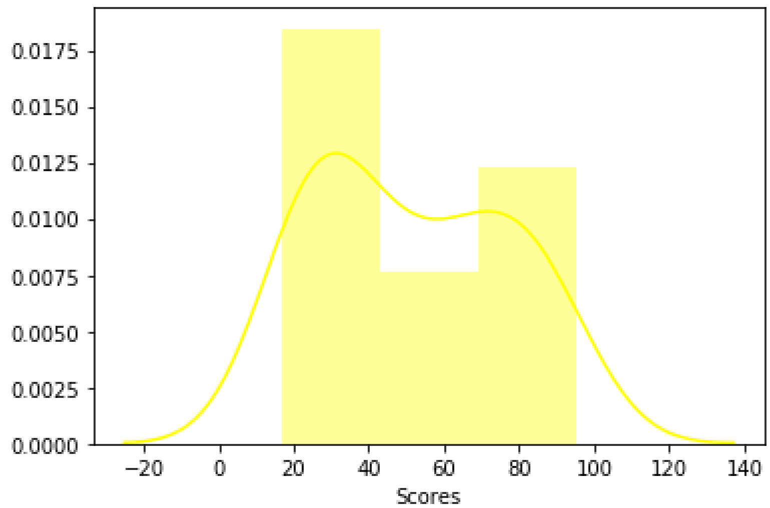
In [50]:
```
# Plotting of Hours data using Distplot.
sns.distplot(DF['Hours'], color='brown')
```

Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0xd04bc62978>



In [52]:
```
# Plotting of Scores data using Distplot.
sns.distplot(DF['Scores'], color='yellow')
```

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0xd04bd38fd0>



Simple Linear Regression Model

Splitting the Data

In [58]:
```
# First we need to divide our data into independent(x) and dependent(y) variable:
x=DF.iloc[:,0:1].values
y=DF.iloc[:, 1].values
```

In [59]:
```
from sklearn.model_selection import train_test_split
```

In [65]:
```
# x_train & y_train for Train the model
# x_test & y_test for Test/ Predict model
X_train, X_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state
```

In [66]:
```
# Check train and test data shapes
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(20, 1)
(5, 1)
(20,)
(5,)
```

Training the Data

In [67]:
```
from sklearn.linear_model import LinearRegression
```

In [70]:
```
# Build Linear Regression Model
Reg = LinearRegression()
```

In [71]:
```
# Train Model
Reg.fit(X_train, y_train)
```

Out[71]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [72]:
```
print("Training completed.")
```

```
Training completed.
```

In [86]:
```
# Using regression coefficients to estimate the coefficients
Coefficient=Reg.coef_
Coefficient
```

Out[86]:
```
array([9.91065648])
```

In [87]:
```
# Using intercept to find mean value of scored when study hours = 0
Intercept=Reg.intercept_
Intercept
```

Out[87]:
```
2.018160041434683
```

Prediction of Score

In [77]:
```
print(X_test)
```

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

In [79]:
```
# Testing Data in hours
y_pred=Reg.predict(X_test)
print(y_pred)
```

```
[16.88414476 33.73226078 75.357018   26.79480124 60.49103328]
```

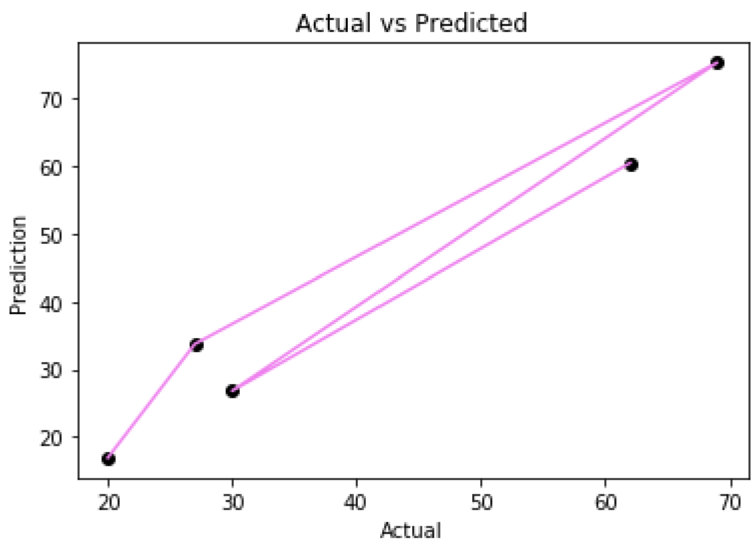
In [81]:
```
# Comparing the values of Actual(y_test) vs Predicted(y_pred)
comparison_dataset=pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
comparison_dataset
```

Out[81]:

	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 [83]:
```
plt.scatter(y_test,y_pred, color='black')
plt.plot(y_test,y_pred, color='violet')
plt.xlabel("Actual")
plt.ylabel("Prediction")
plt.title("Actual vs Predicted")
plt.show()
```



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

In [85]:
```
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 Evaluation

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 [92]:
```
from sklearn import metrics
print('Mean Square Error:', metrics.mean_squared_error(y_test,y_pred))
```

```
Mean Square Error: 21.5987693072174
```

In [95]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,y_pred))
```

```
Mean Absolute Error: 4.183859899002975
```

In [94]:
```
import math
print('Root Mean Square Error:', math.sqrt(metrics.mean_absolute_error(y_test,y_
```

```
Root Mean Square Error: 2.0454485813637495
```

Small value of Mean Absolute Error states that the chances of error or wrong prediction through the model are very less