```
%%html
<marquee style='width: 55%; color: blue;'><b> Machine Learning Lab by Dr.T.Bhaskar  !</b><
```

IMPORT CSV FILES FROM DRIVE INTO GOOGLE-COLAB

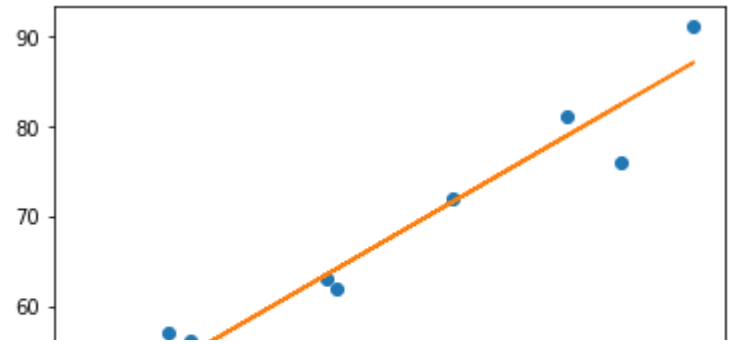
```
#STEP-1: Import Libraries
# Code to read csv file into colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials


#STEP-2: Autheticate E-Mail ID
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)


#STEP-3: Get File from Drive using file-ID

downloaded = drive.CreateFile({'id':'1e10Ynfgrc35FtMl2V5qpzTGyuWF4KQsZ'}) # replace the id
downloaded.GetContentFile('hw.csv')



#STEP-4: Read file as panda dataframe
import pandas as pd
xyz = pd.read_csv('hw.csv')
print(xyz.head(5))
```

```
   height  weight
0     151      63
1     174      81
2     138      56
3     186      91
4     128      47
```

```
#step-5 :Running Actual Program(SLR)
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#Load dataset
dataset=pd.read_csv("hw.csv")
# To display dataset
print(dataset)
```

```
x=dataset.iloc[:,:-1].values
X=dataset.iloc[:,:-1].values
y=dataset.iloc[:,1].values
print(X)
print(y)
#from sklearn subpackage import linear regression model
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X,y)
#To get the slop
regressor.coef_
#To get the y intercept
regressor.intercept_
#To print the equation of line
print("y= "+ str(regressor.coef_) + "X + " + str(regressor.intercept_))

#To get the slop
print("Accuracy:",regressor.score(X,y)*100)
#To plot graph
plt.plot(X,y,'o')
plt.plot(X,regressor.predict(X));
plt.show()
predict_x=int(input('Enter Height:'))
predict_y=(0.67461045*predict_x)-38.45508707607698
plt.scatter(X,y)
plt.scatter(predict_x,predict_y)
plt.xlabel('Enter Height:(Predicted_x)')
plt.ylabel('Enter Weight:(Predicted_y)')
#plotting the Predicted regression line
plt.plot(X,regressor.predict(X),color='green');
plt.show()
```

```
   height  weight
0     151      63
1     174      81
2     138      56
3     186      91
4     128      47
5     136      57
6     179      76
7     163      72
8     152      62
9     131      48
[[151]
 [174]
 [138]
 [186]
 [128]
 [136]
 [179]
 [163]
 [152]
 [131]]
[63 81 56 91 47 57 76 72 62 48]
y= [0.67461045]X + -38.45508707607701
Accuracy: 95.47822477500306
```



IMPLEMENATION OF SIMPLE LINEAR REGRESSION Assignment-1

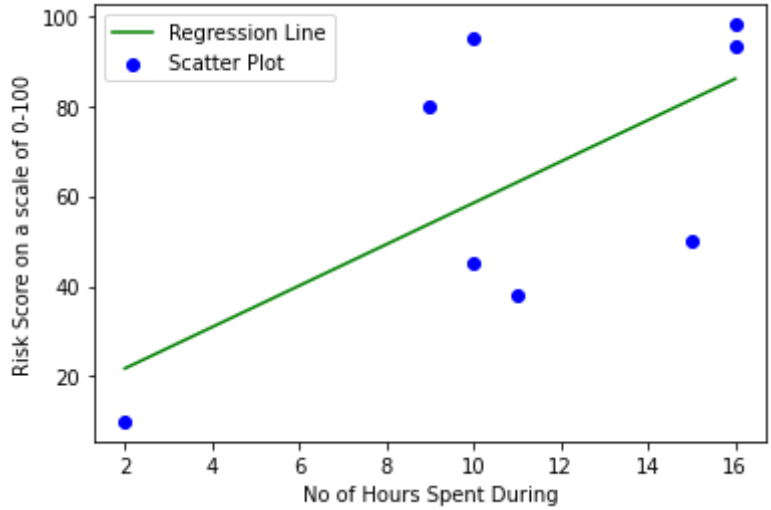
```
downloaded = drive.CreateFile({'id':'1ORTNkZgo6uc5VSYopHxl787fPdrs7-LT'}) # replace the id
downloaded.GetContentFile('SLR-Data.csv')
#https://drive.google.com/file/d/1ORTNkZgo6uc5VSYopHxl787fPdrs7-LT/view?usp=sharing
```

```
#installation of python libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Load dataset
data = pd.read_csv('SLR-Data.csv')
print(data.shape)
print(data.head())
#print(data.describe())

# Collecting X and Y
X = data['No of Hours Spent During(X)'].values
Y = data['Risk Score on a scale of 0-100(Y)'].values
```

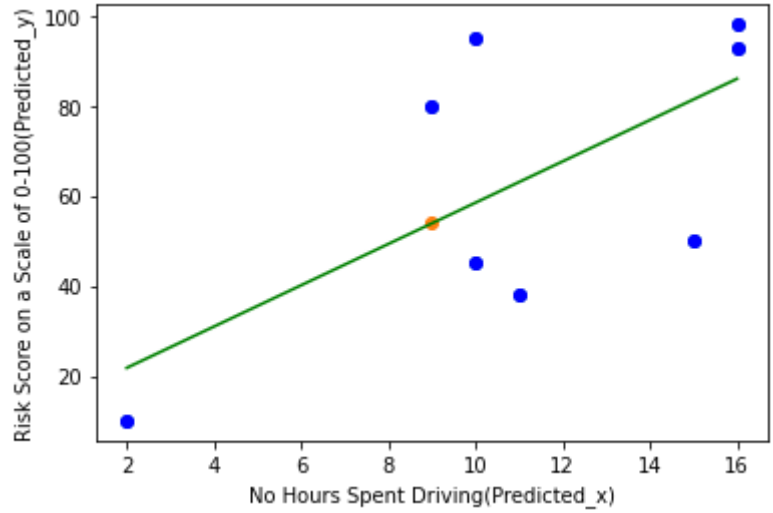
```
# Calculate Mean X and Y
mean_x = np.mean(X)
mean_y = np.mean(Y)
#print(mean_x)
#print(mean_y)
# Total number of values
m = len(X)
# Using the formula to calculate b1(slope) and b0(intercept)
numer = 0
denom = 0
for i in range(m):
    numer += (X[i] - mean_x) * (Y[i] - mean_y)
    denom += (X[i] - mean_x) ** 2
b1 = numer / denom
b0 = mean_y - (b1 * mean_x)

# Print coefficients:b1,b0
print("Slope,Intercept:",b1,b0)

# Plotting Values and Regression Line
max_x = np.max(X)
min_x = np.min(X)

# Calculating line values x and y
x = np.linspace(min_x, max_x)
y = b0 + b1 * x

# Ploting Line
plt.plot(x, y, color='green', label='Regression Line')
# Ploting Scatter Points
plt.scatter(X, Y, c='blue', label='Scatter Plot')
plt.xlabel('No of Hours Spent During')
plt.ylabel('Risk Score on a scale of 0-100')
plt.legend()
plt.show()

#For  Calculating Root Mean Squares Error
rmse = 0
for i in range(m):
    y_pred = b0 + b1 * X[i]
    rmse += (Y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/m)
print("Root Mean Squares Error:",rmse)
# Calculating Accuracy Score
ss_t = 0
ss_r = 0
for i in range(m):
    y_pred = b0 + b1 * X[i]
    ss_t += (Y[i] - mean_y) ** 2
    ss_r += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_r/ss_t)
print("Accuracy:",r2*100)
#predicting a o/p (y) for new value of x
predict_x=int(input('Enter No Hours Spent in Driving:'))
predict_y=(4.58789861*predict_x)+12.584627964022907
```

```
plt.scatter(X,Y)
plt.scatter(predict_x,predict_y)
plt.xlabel('No Hours Spent Driving(Predicted_x)')
plt.ylabel('Risk Score on a Scale of 0-100(Predicted_y)')
 #plotting the regression line
plt.scatter(X, Y, c='blue')
plt.plot(x, y, color='green')
# function to show plot
plt.show()
```

```
(8, 2)
   No of Hours Spent During(X)  Risk Score on a scale of 0-100(Y)
0                           10                                 95
1                            9                                 80
2                            2                                 10
3                           15                                 50
4                           10                                 45
Slope,Intercept: 4.58789860997547 12.584627964022893
```




```
Root Mean Squares Error: 22.759716640449565
Accuracy: 43.709481451010035
Enter No Hours Spent in Driving:9
```

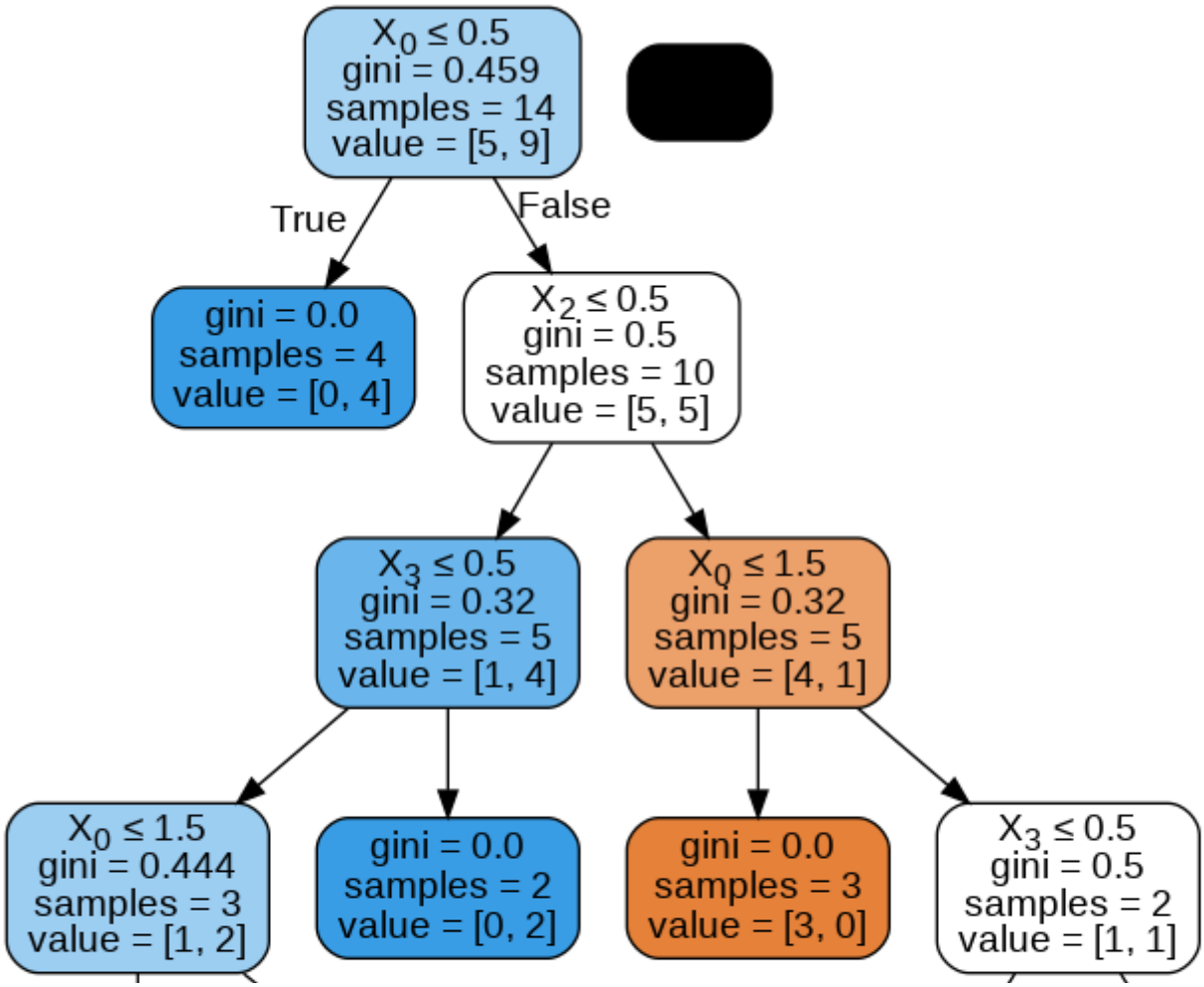


Decision Tree Assignment

```
downloaded = drive.CreateFile({'id':'1jql2mwV15BCFeX52G1PGSCr8Y4jLdn8f'}) # replace the id
downloaded.GetContentFile('DT-Data.csv')
```

```
#import packages
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#reading Dataset
dataset=pd.read_csv("DT-Data.csv")
X=dataset.iloc[:,:-1]
y=dataset.iloc[:,5].values

#Perform Label encoding
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()

X = X.apply(LabelEncoder().fit_transform)
print (X)

from sklearn.tree import DecisionTreeClassifier
regressor=DecisionTreeClassifier()
regressor.fit(X.iloc[:,1:5],y)

#Predict value for the given expression
X_in=np.array([0,1,0,1])

y_pred=regressor.predict([X_in])
print ("Prediction:", y_pred)

from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
# Create DOT data
dot_data = StringIO()

export_graphviz(regressor, out_file=dot_data, filled=True, rounded=True, special_character
# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Decision_Tree.png')
# Show graph
Image(graph.create_png())
```

```
    id  age  income  gender  marital_status
0    0    1       0       1               1
1    1    1       0       1               0
2    2    0       0       1               1
3    3    2       2       1               1
4    4    2       1       0               1
5    5    2       1       0               0
6    6    0       1       0               0
7    7    1       2       1               1
8    8    1       1       0               0
9    9    2       2       0               1
10  10    1       2       0               0
11  11    0       2       1               0
12  12    0       0       0               1
13  13    2       2       1               0
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not h
  "X does not have valid feature names, but"
Prediction: ['Yes']
```



KNN-Assignment

samples = 2 | samples = 1 | samples = 1 | samples =

```
downloaded = drive.CreateFile({'id':'1oikTU46hEkvGW_DeFyWos5_6q3cX6h7B'}) # replace the id
downloaded.GetContentFile('knndata.csv')


#Importing Libraries

import numpy as np
```

```
import pandas as pd

# To split dataset into its attributes and labels.

dataset=pd.read_csv("knndata.csv")
X=dataset.iloc[:,:-1].values
print(X)
Y=dataset.iloc[:,2].values
print(Y)

# Training of KNN Classification Model using trained data

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X,Y)

# Testing  KNN Classification Model using unseen test data

X_test=np.array([6,6])
y_pred = classifier.predict([X_test])
print ('The predicition of classifier is :', y_pred)
classifier = KNeighborsClassifier(n_neighbors=3,weights='distance')
classifier.fit(X,Y)
# predict the class for points(6,6)
X_test=np.array([6,6])
y_pred = classifier.predict([X_test])
print ('The predicition of classifier is :', y_pred)
```

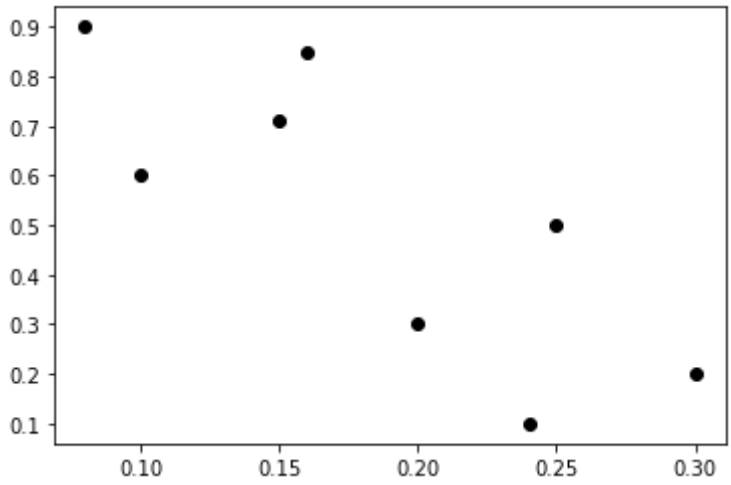
```
[[2 4]
 [4 6]
 [4 4]
 [4 2]
 [6 4]
 [6 2]]
['negative' 'negative' 'postive' 'negative' 'negative' 'positive']
The predicition of classifier is : ['negative']
The predicition of classifier is : ['negative']
```

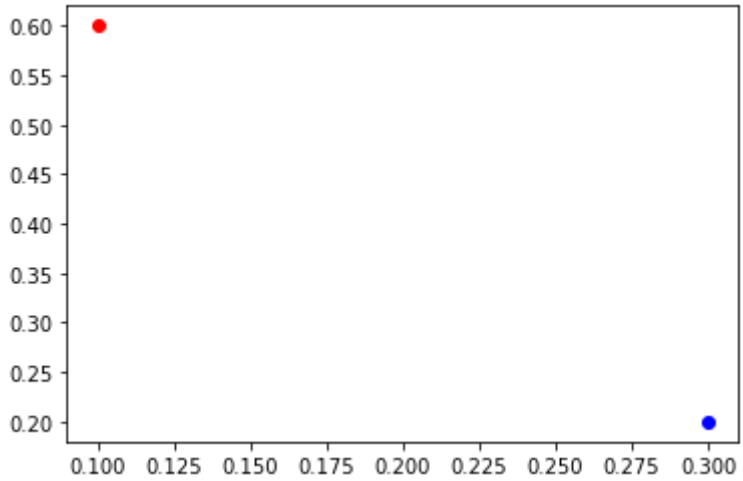
KMeans-Assignment

```
#import packages
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#create dataset using DataFrame
df=pd.DataFrame({'X':[0.1,0.15,0.08,0.16,0.2,0.25,0.24,0.3],
                 'y':[0.6,0.71,0.9,0.85,0.3,0.5,0.1,0.2]})
f1 = df['X'].values
f2 = df['y'].values
X = np.array(list(zip(f1, f2)))
```

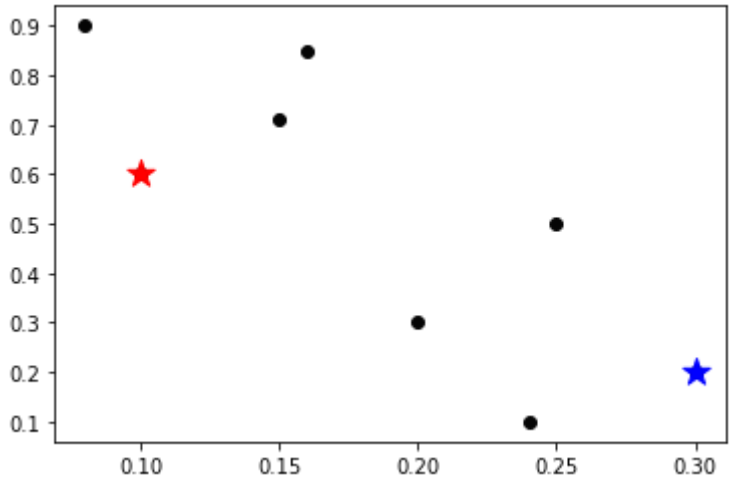
```
print(X)

#centroid points
C_x=np.array([0.1,0.3])
C_y=np.array([0.6,0.2])
centroids=C_x,C_y

#plot the given points
colmap = {1: 'r', 2: 'b'}
plt.scatter(f1, f2, color='k')
plt.show()

#for i in centroids():
plt.scatter(C_x[0],C_y[0], color=colmap[1])
plt.scatter(C_x[1],C_y[1], color=colmap[2])
plt.show()

C = np.array(list((C_x, C_y)), dtype=np.float32)
print (C)

#plot given elements with centroid elements
plt.scatter(f1, f2, c='#050505')
print("point No.6[0.25,0.5] is belongs to blue cluster(cluster no:2)")
plt.scatter(C_x[0], C_y[0], marker='*', s=200, c='r')
plt.scatter(C_x[1], C_y[1], marker='*', s=200, c='b')
plt.show()


#import KMeans class and create object of it
from sklearn.cluster import KMeans
model=KMeans(n_clusters=2,random_state=0)
model.fit(X)
labels=model.labels_
print(labels)

#using labels find population around centroid
count=0
for i in range(len(labels)):
    if (labels[i]==1):
        count=count+1

print('No of population around cluster 2:',count-1)

#Find new centroids
new_centroids = model.cluster_centers_

print('Previous value of m1 and m2 is:')
print('M1==',centroids[0])
print('M1==',centroids[1])

print('Updated value of m1 and m2 is:')
print('M1==',new_centroids[0])
print('M1==',new_centroids[1])
```

```
[[0.1  0.6 ]
 [0.15 0.71]
 [0.08 0.9 ]
 [0.16 0.85]
 [0.2  0.3 ]
 [0.25 0.5 ]
 [0.24 0.1 ]
 [0.3  0.2 ]]
```






```
[[0.1 0.3]
 [0.6 0.2]]
point No.6[0.25,0.5] is belongs to blue cluster(cluster no:2)
```




```
[1 1 1 1 0 0 0 0]
No of population around cluster 2: 3
Previous value of m1 and m2 is:
M1== [0.1 0.3]
M1== [0.6 0.2]
Updated value of m1 and m2 is:
```

```
M1== [0.2475 0.275 ]
M1== [0.1225 0.765 ]
```