Name: Gnanabharathi S

Registration number: 20BRS1186

Course Code: CSE3105

Slot: L39+L40

Lab Report

```
import numpy as np
```

```
a=np.array([1,2,3,4,5])
b=np.array([6,7,8,9,9])
print(a)
print(b)

     [1 2 3 4 5]
     [6 7 8 9 9]
```

```
a=np.zeros((3,3),dtype='float')
print(a)
b=np.eye(3,3)
print(b)
c=np.random.rand(3,3)
print(c)
d=np.random.randint(7,size=(3,3))
print(d)

     [[0. 0. 0.]
      [0. 0. 0.]
      [0. 0. 0.]]
     [[1. 0. 0.]
      [0. 1. 0.]
      [0. 0. 1.]]
     [[0.98948617 0.28642373 0.30562187]
      [0.49363408 0.20372914 0.31223275]
      [0.93795989 0.16960402 0.52299406]]
     [[1 5 2]
      [6 1 5]
      [0 2 3]]
```

```
a=np.arange(1,100,2)
print(a)
b=np.linspace(1,100,5)
print(b)
c=np.array([[1,2,3,4,5],[6,7,8,9,9]])
print(c[0,:])
print(c[0:2,0:2])

     [ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
      49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95
      97 99]
     [  1.    25.75  50.5   75.25 100.  ]
     [1 2 3 4 5]
     [[1 2]
      [6 7]]
```

```
arr11 = np.array([1,2,3,4,5,6,7,8,9])
```

```
mask = np.array([0,1,0,0,1,0,1,0,0],dtype=bool)
print(arr11[mask])
```

```
[2 5 7]
```

```
a=np.array([[1,2,3,4,5],[6,7,8,9,10]])
s=np.sum(a,axis=0)
print(s)
p=np.prod(a,axis=0)
print(p)
mi=np.min(a)
print(mi)
si=np.sin(a)
co=np.cos(a)
ta=np.tan(a)
print(ta)
print(si)
print(co)
lo=np.log(a)
print(lo)
print(np.log(2.7))
```

```
[ 7  9 11 13 15]
[ 6 14 24 36 50]
1
[[ 1.55740772 -2.18503986 -0.14254654  1.15782128 -3.38051501]
 [-0.29100619  0.87144798 -6.79971146 -0.45231566  0.64836083]]
[[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
 [-0.2794155   0.6569866   0.98935825  0.41211849 -0.54402111]]
[[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219]
 [ 0.96017029  0.75390225 -0.14550003 -0.91113026 -0.83907153]]
[[0.         0.69314718 1.09861229 1.38629436 1.60943791]
 [1.79175947 1.94591015 2.07944154 2.19722458 2.30258509]]
0.9932517730102834
```

```
arr17=np.array([1,2])
arr18=np.array([1,1])
result1 = arr17 < arr18
print(result1)
```

```
[False False]
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
```

```
train=pd.read_csv('/content/crime_train.csv')
train=train.drop(["ID"],axis=1)
train.head()
```

	population	householdsize	agePct12t21	agePct12t29	agePct16t24	agePct65up	numbUrl
0	14985	2.56	16.55	34.42	22.54	10.13	
1	30843	2.83	15.45	35.12	18.14	4.70	
2	74991	2.52	10.48	20.43	9.11	20.68	73
3	45061	2.44	10.59	24.97	11.61	16.34	45
4	12863	2.45	12.02	22.51	10.49	18.46	

5 rows × 89 columns

```
train.describe()
```

	population	householdsize	agePct12t21	agePct12t29	agePct16t24	agePct65up
count	1.595000e+03	1595.000000	1595.000000	1595.000000	1595.000000	1595.000000
mean	5.403041e+04	2.702514	14.409141	27.593806	13.944846	11.959335
std	2.195193e+05	0.341554	4.434560	6.136254	5.883211	4.771171
min	1.000500e+04	1.810000	4.680000	9.380000	4.640000	1.660000
25%	1.437350e+04	2.490000	12.240000	24.375000	11.315000	8.985000
50%	2.292200e+04	2.640000	13.640000	26.730000	12.520000	11.830000
75%	4.423950e+04	2.840000	15.345000	29.120000	14.340000	14.470000
max	7.322564e+06	5.280000	54.400000	70.510000	63.620000	52.770000

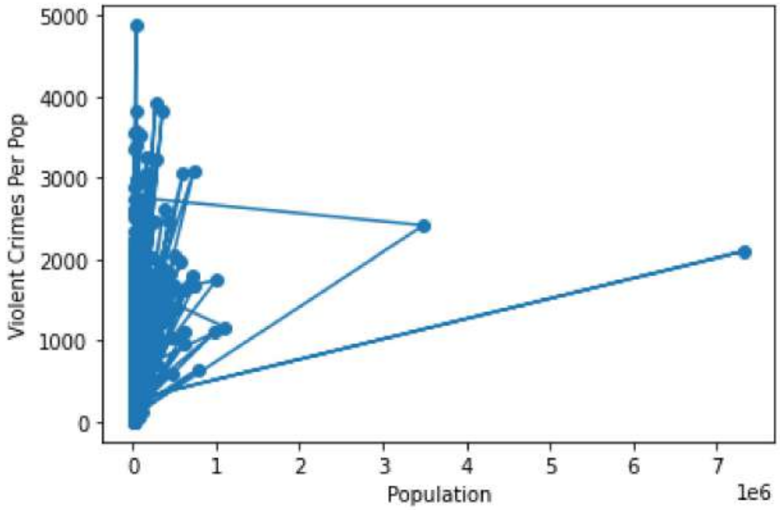
8 rows × 89 columns

```
test=pd.read_csv('/content/crime_test.csv')
test=test.drop(["ID"],axis=1)
test.head()
```

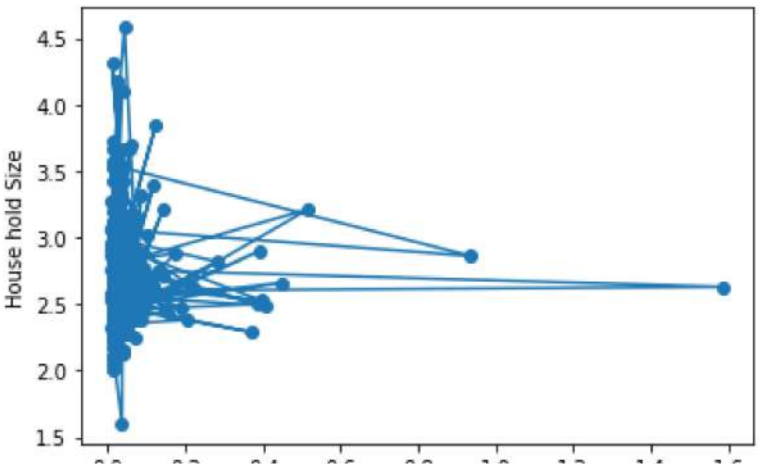
	population	householdsize	agePct12t21	agePct12t29	agePct16t24	agePct65up	numbUrl
0	11874	2.11	10.54	30.87	14.08	8.16	118
1	14143	2.68	21.01	33.35	21.95	14.55	
2	34882	2.32	12.56	21.79	11.29	19.51	348
3	29885	3.53	20.10	34.33	18.31	8.18	298
4	935933	2.86	15.89	30.35	14.98	9.50	9359

5 rows × 88 columns

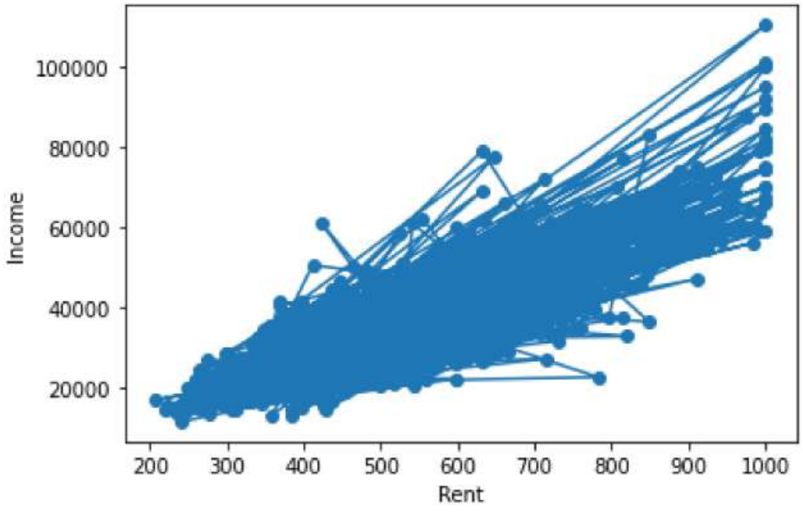
```
plt.scatter(train.population,train.ViolentCrimesPerPop)
plt.plot(train.population,train.ViolentCrimesPerPop)
plt.xlabel('Population')
plt.ylabel('Violent Crimes Per Pop')
plt.show()
```



```
plt.scatter(test.population,test.householdsize)
plt.plot(test.population,test.householdsize)
plt.xlabel('Population')
plt.ylabel('House hold Size')
plt.show()
```



```
plt.scatter(train.MedRent,train.medIncome)
plt.plot(train.MedRent,train.medIncome)
plt.xlabel('Rent')
plt.ylabel('Income')
plt.show()
```



```
from sklearn.linear_model import LinearRegression
```

```
X_T=train.iloc[:,:-1]
Y_T=train.iloc[:,-1]
reg=LinearRegression()
reg.fit(X_T,Y_T)
reg.score(X_T,Y_T)
reg.predict(X_T)
```

```
array([ 259.87557878,  567.1237    ,  413.71565939, ..., 1126.70024816,
       1056.1260383 ,  339.0973182 ])
```

```
X_T=test.iloc[:,:-1]
Y_T=test.iloc[:,-1]
reg=LinearRegression()
reg.fit(X_T,Y_T)
reg.score(X_T,Y_T)
reg.predict(X_T[0:20])
```

```
array([85.60254379, 89.9995928 , 90.81934825, 91.60551927, 90.82659769,
       75.90451551, 81.30849646, 89.18677298, 90.2972196 , 86.0989516 ,
       86.02252353, 90.80515326, 93.97619802, 78.51695206, 88.20714127,
       83.96128167, 95.14669323, 70.25666301, 89.43291127, 88.85064114])
```

```
from scipy import stats
slope, intercept, r, p, std_err = stats.linregress(train.population,train.ViolentCrimesPerPop
print('Slope = ',slope)
print('Intercept = ',slope)
def myfunc(x):
  return slope * x + intercept

print("Enter a random population to get predicted Violent Crime Per Pop")
pop=int(input())
crime = myfunc(pop)
print("When the population = ",pop," Violent Crime Per Pop = ", crime)
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error([30843,742.54],[pop,crime]))
print(rmse)
```

```
Slope =  0.0006080841195374458
Intercept =  0.0006080841195374458
Enter a random population to get predicted Violent Crime Per Pop
20000
When the population =  20000  Violent Crime Per Pop =  570.8338248557552
7668.12010895052
```

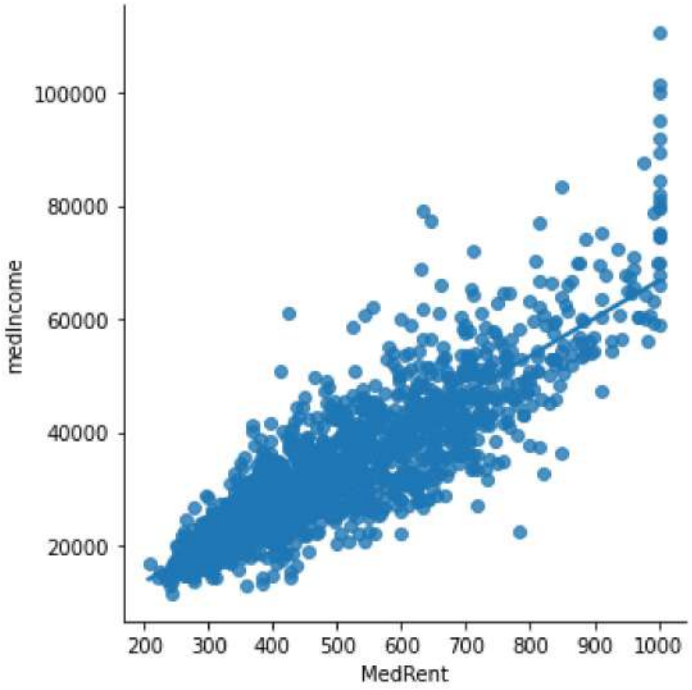
```
from google.colab import drive
drive.mount('/content/drive')
```

```
from scipy import stats
slope, intercept, r, p, std_err = stats.linregress(train.MedRent,train.medIncome)
print('Slope = ',slope)
print('Intercept = ',slope)
def myfunc(x):
  return slope * x + intercept

print("Enter a random medRent to get predicted medIncome")
pop=int(input())
crime = myfunc(pop)
print("When the medRent = ",pop," medIncome = ", crime)

import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error([670,35545],[pop,crime]))
print(rmse)
```

```
Slope =  66.50348186436204
Intercept =  66.50348186436204
Enter a random medRent to get predicted medIncome
2000
When the medRent =  2000  medIncome =  133274.1225228662
69111.32428585563
```

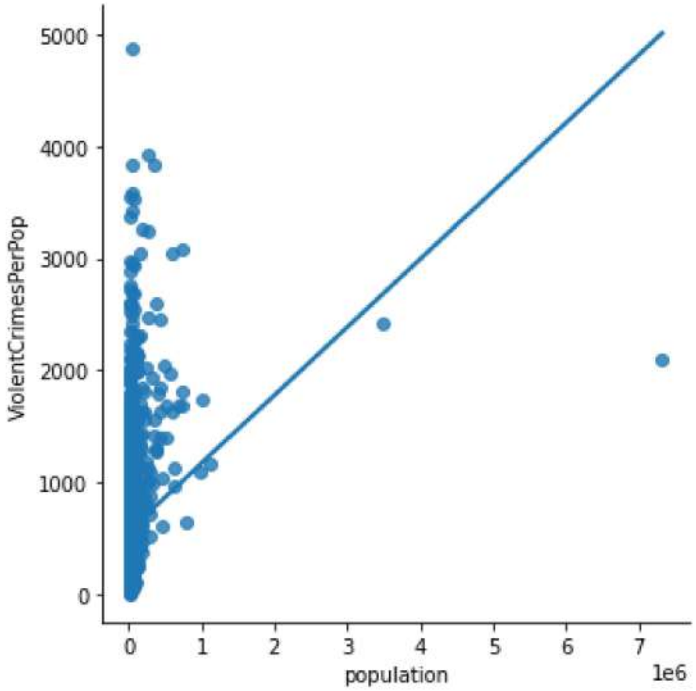
```
import seaborn as sb

# load data
df = train

# use lmplot
sb.lmplot(x = "MedRent",
            y = "medIncome",
            ci = None,
            data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f39c4f8b490>
```



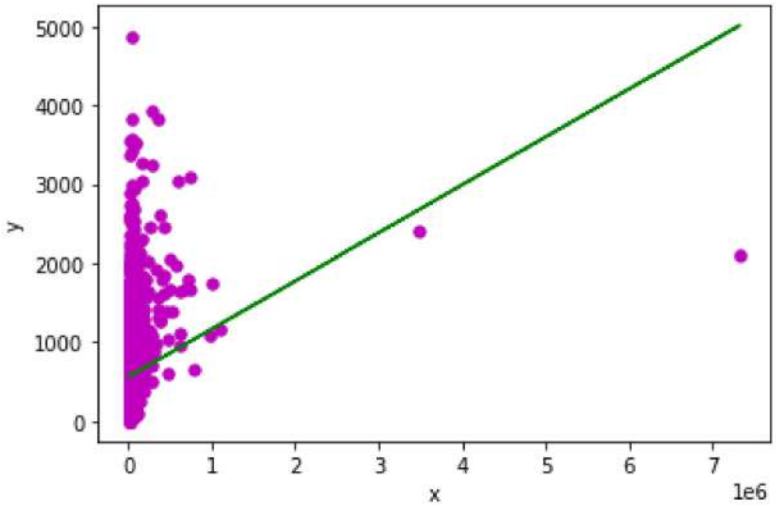
```
import seaborn as sb

# load data
df = train

# use lmplot
sb.lmplot(x = "population",
            y = "ViolentCrimesPerPop",
            ci = None,
            data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f39c441e210>
```

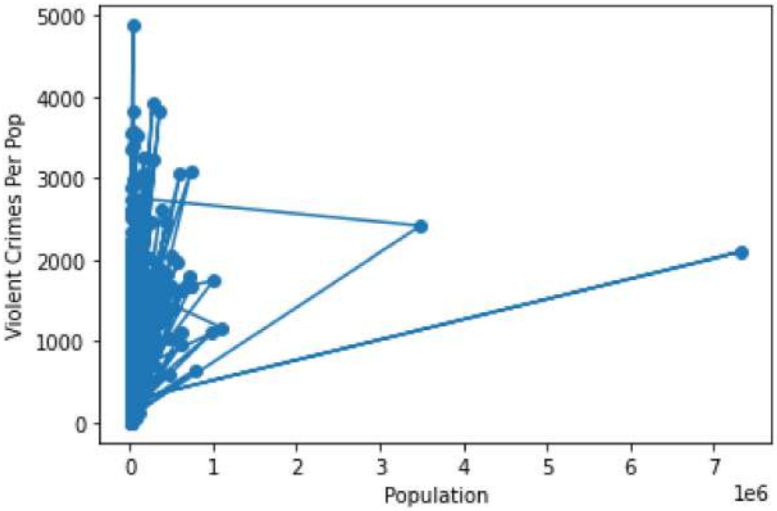


```
def estimate_coef(x, y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    y_pred = b[0] + b[1]*x

    plt.plot(x, y_pred, color = "g")

    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

x=train.population
y=train.ViolentCrimesPerPop
b = estimate_coef(x,y)
print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))
plot_regression_line(x, y, b)
```

```
Estimated coefficients:
b_0 = 558.6721424650065
b_1 = 0.0006080841195374455
```



```
plt.scatter(train.population,train.ViolentCrimesPerPop)
plt.plot(train.population,train.ViolentCrimesPerPop)
plt.xlabel('Population')
plt.ylabel('Violent Crimes Per Pop')
plt.show()
```



```
import math

from sklearn.metrics import mean_squared_error

rmse = math.sqrt(mean_squared_error([100],[200]))
print(rmse)
```

```
100.0
```

```
from scipy import stats
slope, intercept, r, p, std_err = stats.linregress(train.population,train.ViolentCrimesPerPop
print('Slope = ',slope)
print('Intercept = ',slope)
def myfunc(x):
  return slope * x + intercept

crime = myfunc(14985)
print("When the population = ",14985," Violent Crime Per Pop = ", crime)
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error([428.64],[crime]))
print(rmse)
```

```
    Slope =  0.0006080841195374458
    Intercept =  0.0006080841195374458
    When the population =  14985  Violent Crime Per Pop =  567.784282996275
    98.3898660700061
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as py
```

```
tennis=pd.read_csv('/content/tennis.csv')
tennis.head()
tennis.tail()
```

	day	outlook	temp	humidity	wind	play
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
13	D14	Rain	Mild	High	Strong	No

```
X=tennis.iloc[:,1:5].values
y=tennis.iloc[:,-1].values
print(X)
print(y)
```

```
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
['No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes'
 'No']
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)
print(y)
```

```
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import make_scorer, accuracy_score,precision_score,classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import CategoricalNB


X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.5,random_state=0)
print(X_train)
```

```
[['Sunny' 'Hot' 'High' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']]
```

```
tennis=pd.read_csv('/content/tennis.csv')
tennis.head()
tennis.tail()
X=tennis.iloc[:,1:5].values
y=tennis.iloc[:,-1].values
print(X)
print(y)
outlook1={'Sunny':1, 'Overcast':2, 'Rain':3}
tennis.outlook=tennis.outlook.map(outlook1)

temp1={'Hot':1, 'Mild':2, 'Cool':3}
tennis.temp=tennis.temp.map(temp1)

humid1={'Normal':1, 'High':2}
tennis.humidity=tennis.humidity.map(humid1)

wind1={'Weak':1, 'Strong':0}
tennis.wind=tennis.wind.map(wind1)

play1={'Yes':1, 'No':0}

tennis.play=tennis.play.map(play1)
x=tennis.iloc[:,1:5].values
y=tennis.iloc[:,-1].values
print(x)
print(y)
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.3,random_state=0)
print('xtrain = ',x_train)
print('xtest = ',x_test)
print('ytrain = ',y_train)
print('ytest = ',y_test)
gnb=GaussianNB()
gnb.fit(x_train,y_train)
y_pred=gnb.predict(x_test)
print('Confusion matrix = ',confusion_matrix(y_test,y_pred))
```

```
accuracy_nb=(accuracy_score(y_test,y_pred)*100,2)
acc=(gnb.score(x_train,y_train)*100,2)
accuracy=accuracy_score(y_pred,y_test)
precision=precision_score(y_test,y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):", acc[0])
print(accuracy_score(y_test,y_pred)*100)
print(recall_score(y_test,y_pred))
```

```
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
['No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes'
 'No']
[[1 1 2 1]
 [1 1 2 0]
 [2 1 2 1]
 [3 2 2 1]
 [3 3 1 1]
 [3 3 1 0]
 [2 3 1 0]
 [1 2 2 1]
 [1 3 1 1]
 [3 2 1 1]
 [1 2 1 0]
 [2 2 2 0]
 [2 1 1 1]
 [3 2 2 0]]
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
xtrain =  [[3 2 2 1]
 [1 1 2 1]
 [3 3 1 0]
 [2 1 1 1]]
xtest =  [[1 3 1 1]
 [2 3 1 0]
 [3 3 1 1]
 [2 2 2 0]
 [2 1 2 1]
 [3 2 2 0]
 [3 2 1 1]
 [1 1 2 0]
 [1 2 2 1]
 [1 2 1 0]]
ytrain =  [1 0 0 1]
```

```
ytest =  [1 1 1 1 1 0 1 0 0 1]
Confusion matrix =  [[2 1]
 [3 4]]
Gaussian Naive Bayes model accuracy(in %): 75.0
60.0
0.5714285714285714
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
```

```
data_train = pd.read_csv("/content/play_tennis_train.csv")
data_train.head()
```

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes

```
x_train = data_train[['outlook','temp','humidity','wind']]
print(x_train)
y_train = data_train['play']
print(y_train)
```

```
    outlook  temp humidity    wind
0     Sunny   Hot     High    Weak
1     Sunny   Hot     High  Strong
2  Overcast   Hot     High    Weak
3      Rain  Mild     High    Weak
4      Rain  Cool   Normal    Weak
5      Rain  Cool   Normal  Strong
6  Overcast  Cool   Normal  Strong
7     Sunny  Mild     High    Weak
8     Sunny  Cool   Normal    Weak
9      Rain  Mild   Normal    Weak
0     No
1     No
2    Yes
3    Yes
4    Yes
5     No
6    Yes
7     No
8    Yes
9    Yes
Name: play, dtype: object
```

```
data_test = pd.read_csv("/content/play_tennis_test.csv")
data_test.head()
```

	day	outlook	temp	humidity	wind	play
0	D11	Sunny	Mild	Normal	Strong	Yes
1	D12	Overcast	Mild	High	Strong	Yes
2	D13	Overcast	Hot	Normal	Weak	Yes
3	D14	Rain	Mild	High	Strong	No

```
x_test = data_test[['outlook','temp','humidity','wind']]
print(x_test)
y_test = data_test['play']
```

```
    outlook  temp humidity    wind
0     Sunny  Mild   Normal  Strong
1  Overcast  Mild     High  Strong
2  Overcast   Hot   Normal    Weak
3      Rain  Mild     High  Strong
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

```
le = LabelEncoder()
le.fit(y_train)
y_train_l=le.transform(y_train)
print(y_train_l)
le.fit(y_test)
y_test_l = le.transform(y_test)
print(y_test_l)
```

```
[0 0 1 1 1 0 1 0 1 1]
[1 1 1 0]
```

```
le.fit(x_train['outlook'])
print(x_train['outlook'])

x0_l=le.transform(x_train['outlook'])
x0_l1=le.transform(x_test['outlook'])
print(x0_l)
le.fit(x_train['temp'])
print(list(le.classes_))
x1_l=le.transform(x_train['temp'])
x1_l1=le.transform(x_test['temp'])
le.fit(x_train['humidity'])
x2_l=le.transform(x_train['humidity'])
x2_l1=le.transform(x_test['humidity'])
```

```
le.fit(x_train['wind'])
x3_l=le.transform(x_train['wind'])
x3_l1=le.transform(x_test['wind'])
x_train_l = np.array([x0_l,x1_l,x2_l,x3_l])
x_test_l = np.array([x0_l1,x1_l1,x2_l1,x3_l1])
x_test_l = x_test_l.transpose()
print("X test data:",x_test_l)
x_train_l = x_train_l.transpose()
print("X train data",x_train_l)
```

```
0       Sunny
1       Sunny
2    Overcast
3        Rain
4        Rain
5        Rain
6    Overcast
7       Sunny
8       Sunny
9        Rain
Name: outlook, dtype: object
[2 2 0 1 1 1 0 2 2 1]
['Cool', 'Hot', 'Mild']
X test data: [[2 2 1 0]
 [0 2 0 0]
 [0 1 1 1]
 [1 2 0 0]]
X train data [[2 1 0 1]
 [2 1 0 0]
 [0 1 0 1]
 [1 2 0 1]
 [1 0 1 1]
 [1 0 1 0]
 [0 0 1 0]
 [2 2 0 1]
 [2 0 1 1]
 [1 2 1 1]]
```

```
gnd = GaussianNB()
```

```
gnd.fit(x_train_l,y_train_l)
```

```
GaussianNB()
```

```
y_pred=gnd.predict(x_test_l)
print(y_pred)
print(y_test_l)
accuracy_score(y_test_l,y_pred)*100
```

```
[0 1 1 0]
[1 1 1 0]
75.0
```

```
confusion_matrix(y_test_l,y_pred)
```

```
array([[1, 0],
       [1, 2]])
```

```
recall_score(y_test_l,y_pred)
```

```
0.6666666666666666
```

```
f1_score(y_test_l, y_pred, average='macro')
```

```
0.7333333333333334
```

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import plot_tree
from sklearn.metrics import classification_report
import graphviz
```

```
data=pd.read_csv('/content/Comp.csv')
data.head()
```

	age	Income	Student	Credit_Rating	Buys_Computer
0	<=30	high	no	fair	no
1	<=30	high	no	excellent	no
2	31...40	high	no	fair	yes
3	>40	medium	no	fair	yes
4	>40	low	yes	fair	yes

```
data=data.replace(['<=30','31...40','>40'],[1,2,3])
data.head()
data=data.replace(['high','medium','low'],[1,2,3])
data.head()
data=data.replace(['no','yes'],[1,2])
data.head()
data=data.replace(['fair','excellent'],[1,2])
data.head()
#X=data.drop(columns=['Outcome'])
#Y=data['Outcome']
#X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.9, random_state =0)
```

	age	Income	Student	Credit_Rating	Buys_Computer
0	1	1	1	1	1
1	1	1	1	2	1
2	2	1	1	1	2
3	3	2	1	1	2
4	3	3	2	1	2

```
X=data.drop(columns=['Buys_Computer'])
```

```
Y=data['Buys_Computer']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.9, random_state =0)
print("Entropy")
model = DecisionTreeClassifier(criterion = "entropy")
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Predicted values:")
print(y_pred)
print("Confusion Matrix: ",confusion_matrix(y_test, y_pred))
print("Accuracy : ",accuracy_score(y_test,y_pred)*100)
print("Report : ",classification_report(y_test, y_pred))

dtree = DecisionTreeClassifier()
dtree = dtree.fit(X_test, y_test)
features = ['age', 'Income', 'Student', 'Credit_Rating','Buys_Computer']
tree.plot_tree(dtree, feature_names=features)
```

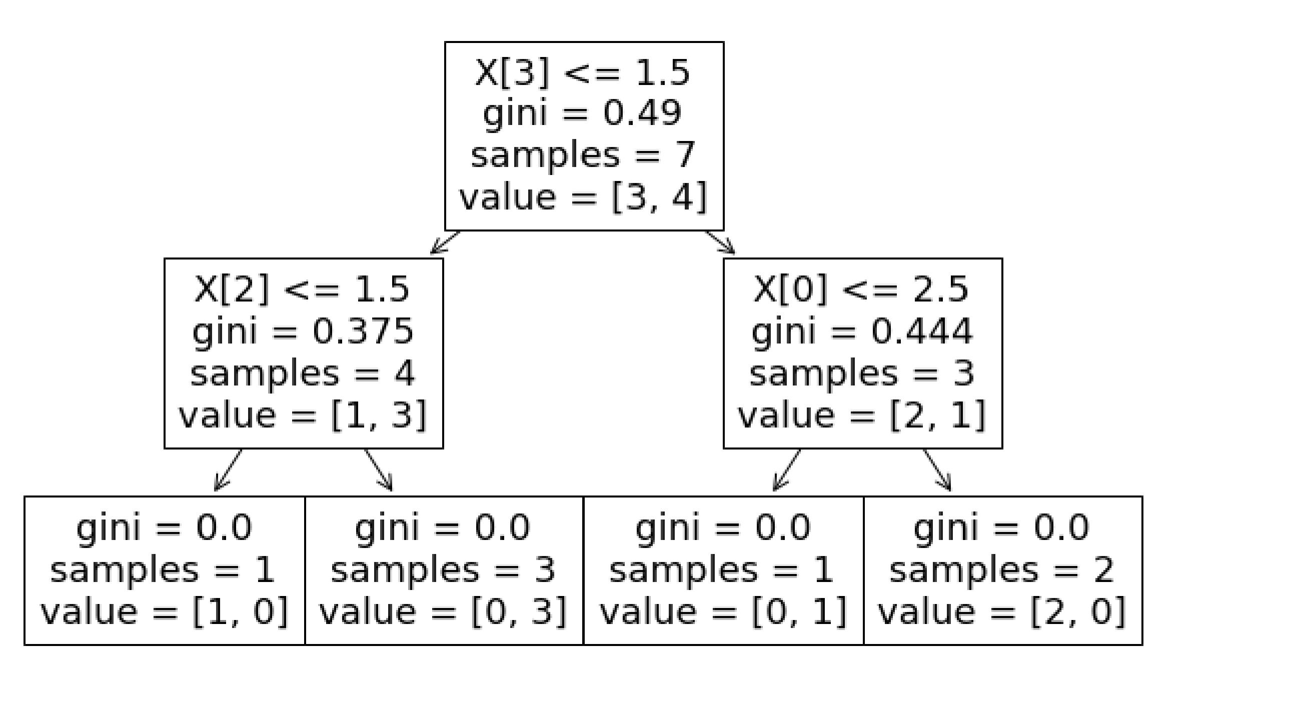
```
Entropy
Predicted values:
[2 2 2 2 2 2 2 2 2 2 2 2 2]
Confusion Matrix:  [[0 5]
 [0 8]]
Accuracy :  61.53846153846154
Report :                precision    recall  f1-score   support
```

```
X=data.drop(columns=['Buys_Computer'])
Y=data['Buys_Computer']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.5, random_state =1)
print("Gini")
model=DecisionTreeClassifier(criterion = "gini")
model=model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Predicted values:")
print(y_pred)
print("Confusion Matrix: ",confusion_matrix(y_test, y_pred))
print("Accuracy : ",accuracy_score(y_test,y_pred)*100)
print("Report : ",classification_report(y_test, y_pred))
```

```
Gini
Predicted values:
[2 1 2 2 1 2 1]
Confusion Matrix:  [[2 0]
 [1 4]]
Accuracy :  85.71428571428571
Report :                precision    recall  f1-score   support

           1       0.67      1.00      0.80         2
           2       1.00      0.80      0.89         5

    accuracy                           0.86         7
   macro avg       0.83      0.90      0.84         7
weighted avg       0.90      0.86      0.86         7
```

```
2]'),
```

```
plt.figure(figsize=(10,6))
plot_tree(model)
plt.show()
```


X[3] <= 1.5
gini = 0.49
samples = 7
value = [3, 4]
X[2] <= 1.5
gini = 0.375
samples = 4
value = [1, 3]
X[0] <= 2.5
gini = 0.444
samples = 3
value = [2, 1]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 3
value = [0, 3]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 2
value = [2, 0]

Name: Gnanabharathi

Registration number: 20BRS1186

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import tree

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import graphviz
```

```
data=pd.read_csv('/content/Result.csv')
data.head()
```

	Unnamed: 0	CAT1	CAT2	DA1	DA2	DA3	FAT	Outcome
0	0	12.25	6.90	8	5	9	25.4	Pass
1	1	11.00	7.65	10	9	6	9.0	Fail
2	2	15.00	10.50	10	6	6	24.4	Pass
3	3	9.50	11.10	8	10	7	29.6	Pass
4	4	7.50	5.40	9	9	8	24.4	Pass

```
data=data.replace(['Pass','Fail'],[1,0])
X=data.drop(columns=['Outcome'])
Y=data['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.9, random_state =0)
```

```
print("Gini")
model=DecisionTreeClassifier(criterion = "gini")
model=model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Predicted values:")
print(y_pred)
```

```
Gini
Predicted values:
[1 1 1 1 1 1 1 1 1 1]
```

```
print("Confusion Matrix: ",confusion_matrix(y_test, y_pred))
print("Accuracy : ",accuracy_score(y_test,y_pred)*100)
print("Report : ",classification_report(y_test, y_pred))
```

```
Confusion Matrix:  [[0 2]
 [0 8]]
Accuracy :  80.0
```

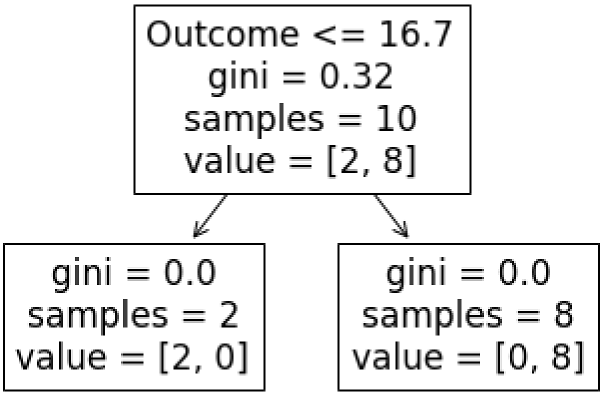
```
Report :                precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.80      1.00      0.89         8

    accuracy                           0.80        10
   macro avg       0.40      0.50      0.44        10
weighted avg       0.64      0.80      0.71        10

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
```

```
print("Entropy")
ent = DecisionTreeClassifier(criterion = "entropy")
ent.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Predicted values:")
print(y_pred)
print("Confusion Matrix: ",confusion_matrix(y_test, y_pred))
print("Accuracy : ",accuracy_score(y_test,y_pred)*100)
print("Report : ",classification_report(y_test, y_pred))
```

```
Entropy
Predicted values:
[1 1 1 1 1 1 1 1 1 1]
Confusion Matrix:  [[0 2]
 [0 8]]
Accuracy :  80.0
Report :                precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.80      1.00      0.89         8

    accuracy                           0.80        10
   macro avg       0.40      0.50      0.44        10
weighted avg       0.64      0.80      0.71        10

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
```

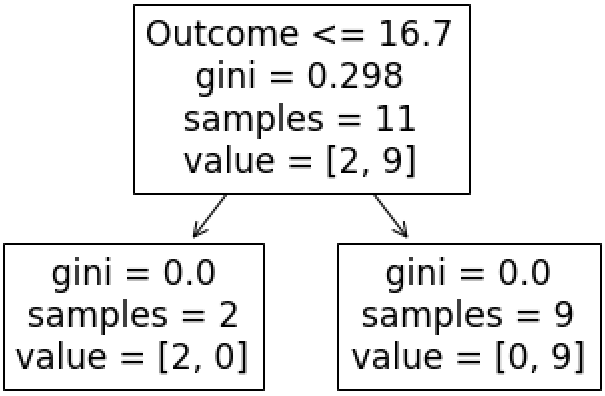
```
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X_test, y_test)
features = ['CAT1', 'CAT2', 'DA1', 'DA2','DA3','FAT','Outcome']
tree.plot_tree(dtree, feature_names=features)
```

```
[Text(0.5, 0.75, 'Outcome <= 16.7\ngini = 0.32\nsamples = 10\nvalue = [2, 8]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]')]
```



```
dtree = dtree.fit(X,Y)
features = ['CAT1', 'CAT2', 'DA1', 'DA2','DA3','FAT','Outcome']
tree.plot_tree(dtree, feature_names=features)
```

```
[Text(0.5, 0.75, 'Outcome <= 16.7\ngini = 0.298\nsamples = 11\nvalue = [2, 9]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]')]
```



```
import numpy as np
import pandas as pd
import seaborn as sea
import matplotlib.pyplot as plt
```

```
data = pd.read_csv('/content/iris.csv')
data.head()
```

	sepallength	sepalwidth	petallength	petalwidth	class1
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Double-click (or enter) to edit

```
data.describe()
```

	sepallength	sepalwidth	petallength	petalwidth
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
data['class1'].value_counts()
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: class1, dtype: int64
```

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

```
sepallength    0
sepalwidth     0
petallength    0
```

```
petalwidth     0
class1         0
dtype: int64
```

```
data.corr()
```

	sepallength	sepalwidth	petallength	petalwidth
sepallength	1.000000	-0.109369	0.871754	0.817954
sepalwidth	-0.109369	1.000000	-0.420516	-0.356544
petallength	0.871754	-0.420516	1.000000	0.962757
petalwidth	0.817954	-0.356544	0.962757	1.000000

```
from sklearn.preprocessing import LabelEncoder
enc=LabelEncoder()
data['class1']=enc.fit_transform(data['class1'])
print(data['class1'])
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: class1, Length: 150, dtype: int64
```

```
data.head()
```

	sepallength	sepalwidth	petallength	petalwidth	class1
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
from sklearn.model_selection import train_test_split
```

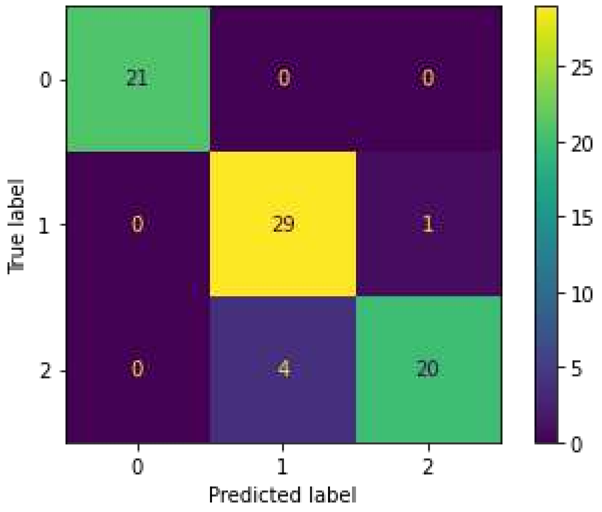
```
X=data.drop(columns=['class1'])
Y=data['class1']
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.5,random_state=0)
```

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model=model.fit(X_train,Y_train)
pred=model.predict(X_test)
print("Accuracy = ",model.score(X_test,Y_test)*100)
```

```
Accuracy =  93.33333333333333
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
from sklearn.metrics import classification_report, plot_confusion_matrix,precision_score,recal
print(classification_report(Y_test,pred))
plot_confusion_matrix(model,X_test,Y_test)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       0.88      0.97      0.92        30
           2       0.95      0.83      0.89        24

    accuracy                           0.93        75
   macro avg       0.94      0.93      0.94        75
weighted avg       0.94      0.93      0.93        75

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd9034fa4d0>
```



```
from sklearn.metrics import confusion_matrix
print('Confusion Matrix = ',confusion_matrix(Y_test,pred))
print('Precision = ',(precision_score(Y_test,pred,average='micro')))
```

```
Confusion Matrix =  [[21  0  0]
 [ 0 29  1]
 [ 0  4 20]]
Precision =  0.9333333333333333
```

```
pip install cv
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Collecting cv
  Downloading cv-1.0.0-py3-none-any.whl (7.3 kB)
Installing collected packages: cv
Successfully installed cv-1.0.0
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

Colab paid products - Cancel contracts here

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
```

```
train_data = pd.read_csv('/content/wheet_train.csv')
tr = train_data.drop(['ID'],axis=1)
tr.head()
```

	area	perimeter	compactness	kernelLength	kernelWidth	asymmetryCoefficient	ke
0	18.59	16.05	0.9066	6.037	3.860	6.001	
1	11.18	12.72	0.8680	5.009	2.810	4.051	
2	15.99	14.89	0.9064	5.363	3.582	3.336	
3	15.38	14.90	0.8706	5.884	3.268	4.462	
4	19.15	16.45	0.8890	6.245	3.815	3.084	

```
y = tr['Type']
X = tr.drop(['Type'], axis=1)
```

```
test_data = pd.read_csv('/content/wheet_test.csv')
test_data.head()
```

	ID	area	perimeter	compactness	kernelLength	kernelWidth	asymmetryCoefficient
0	1	18.85	16.17	0.9056	6.152	3.806	2.843
1	2	11.34	12.87	0.8596	5.053	2.849	3.347
2	3	14.86	14.67	0.8676	5.678	3.258	2.129
3	4	12.67	13.32	0.8977	4.984	3.135	2.300
4	5	11.82	13.40	0.8274	5.314	2.777	4.471

```
tst = test_data.drop(['ID'],axis=1)
```

```
log_reg = LogisticRegression()
log_reg = log_reg.fit(X,y)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
y_test = log_reg.predict(tst)
print(y_test)
```

```
[2 3 1 1 3 2 1 2 1 2 3 1 2 2 3 3 2 2 3 1 3 1 3 2 3 1 3 1 2 2 1 2 2 1 3 2 2
 2 2 2 3 3 3 3 1 2 3 1 3 2 3 1 1 3 3 1 3 2 2 1 1 2 2 2 1 1 2 3 1 1]
```

```
id = test_data['ID']
```

```
import csv
submission = open("log_regr.csv","w")
sub_file = csv.writer(submission)
sub_file.writerow(['ID','Type'])
for i in range(0,len(y_test)):
    sub_file.writerow([str(id[i]),str(y_test[i])])

submission.close()
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
```

```
data=pd.read_csv('/content/Train Knn.csv')
data.dropna(inplace=True)
data.drop(columns=['loan_id'],inplace=True)
```

```
data.shape
```

```
(6755, 9)
```

```
data.columns
```

```
Index(['age', 'education', 'proof_submitted', 'loan_amount', 'asset_cost',
       'no_of_loans', 'no_of_curr_loans', 'last_delinq_none', 'loan_default'],
      dtype='object')
```

```
data.head(10)
```

	age	education	proof_submitted	loan_amount	asset_cost	no_of_loans	no_of_curr
0	27	1.0	Aadhar	504264	820920	2	
1	48	1.0	Aadhar	728556	831444	6	
2	30	2.0	VoterID	642936	826092	0	
3	28	1.0	Aadhar	746556	930924	0	
4	29	1.0	Aadhar	1139880	1902000	0	
5	34	2.0	Aadhar	779784	902040	0	
6	27	2.0	Aadhar	449268	847896	0	
7	27	2.0	Aadhar	582036	905604	0	
8	30	1.0	Aadhar	712956	866292	0	
9	46	2.0	Aadhar	554988	761724	3	

```
#look_up_fruit=dict(zip(data.fruit_label.unique(), data.fruit_name.unique()))
```

```
#look_up_fruit
```

```
label_encoder = preprocessing.LabelEncoder()
data['proof_submitted']= label_encoder.fit_transform(data['proof_submitted'])
data.head()
```

	age	education	proof_submitted	loan_amount	asset_cost	no_of_loans	no_of_curr
0	27	1.0	0	504264	820920	2	
1	48	1.0	0	728556	831444	6	
2	30	2.0	4	642936	826092	0	
3	28	1.0	0	746556	930924	0	
4	29	1.0	0	1139880	1902000	0	

```
X=data[['age','education','proof_submitted','loan_amount','asset_cost','no_of_loans','no_of_
y=data['loan_default']
```

Double-click (or enter) to edit

```
X_train, X_test, y_train, y_test=train_test_split(X, y, random_state=0)
```

```
knn=KNeighborsClassifier(n_neighbors=5)
```

```
"""if(np.any(np.isnan(X)) or np.all(np.isfinite(X)) or np.any(np.isnan(y)) or np.all(np.isfi
  print()
else:
  knn.fit(X_train, y_train)"""
knn.fit(X_train, y_train)
```

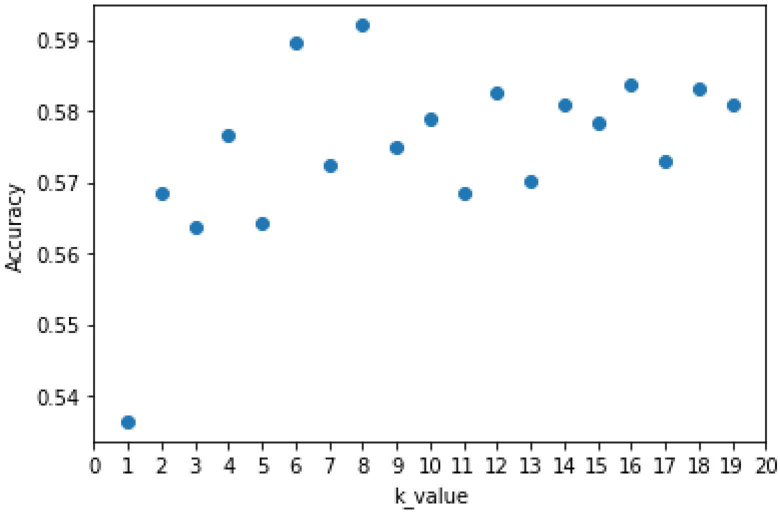
```
KNeighborsClassifier()
```

```
tdata=pd.read_csv('/content/Test Knn.csv')
tdata.dropna(inplace=True)
tdata.drop(columns=['loan_id'],inplace=True)
tdata['proof_submitted']= label_encoder.fit_transform(tdata['proof_submitted'])
Xt=tdata[['age','education','proof_submitted','loan_amount','asset_cost','no_of_loans','no_o
knn.fit(X_train,X_test)
y_predict=knn.predict(Xt)
#y_correct=np.array(y_test)
#print(np.concatenate((y_predict.reshape(len(y_predict), 1), y_correct.reshape(len(y_correct
```

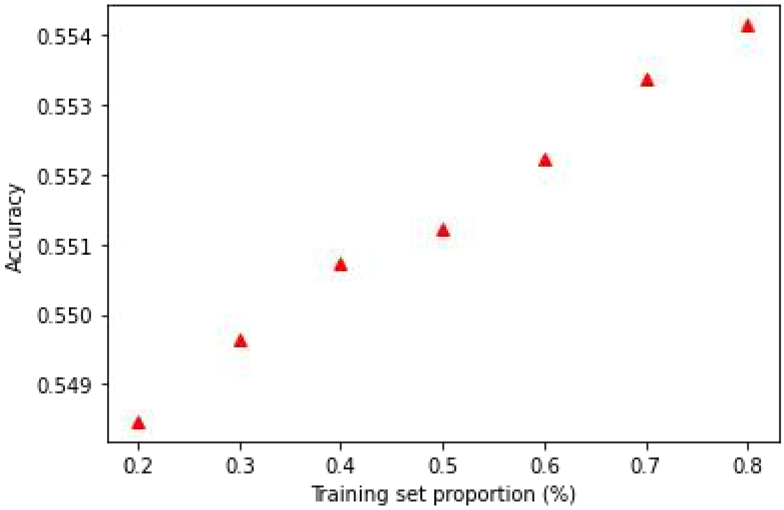
```
knn.score(X_train, y_train)
```

```
0.7090406632451638
```

```
k_range = range(1,20)
scores = []
for k in k_range:
 knn = KNeighborsClassifier(n_neighbors = k)
 knn.fit(X_train, y_train)
 scores.append(knn.score(X_test, y_test))
plt.figure()
plt.xlabel('k_value')
plt.ylabel('Accuracy')
plt.scatter(k_range, scores)
plt.xticks(range(0,21));

```



```
t = [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
knn = KNeighborsClassifier(n_neighbors = 3)
plt.figure()
for split in t:
  scores = []
  for i in range(1,1000):
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = split)
    knn.fit(X_train, y_train)
    scores.append(knn.score(X_test, y_test))
  plt.plot(split, np.mean(scores), 'r^')
plt.xlabel('Training set proportion (%)')
plt.ylabel('Accuracy');
```



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

```
dataset = pd.read_csv('/content/IRIS.csv')
dataset.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values
```

```
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)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
```

```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

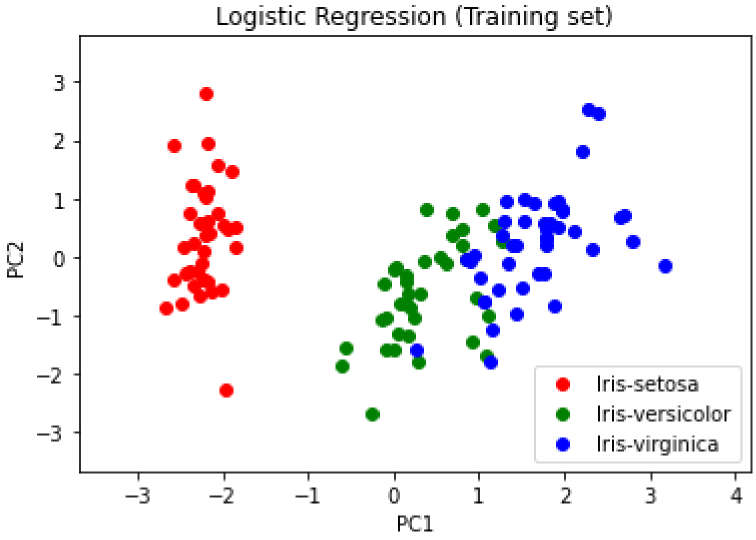
```
LogisticRegression(random_state=0)
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```

```
from matplotlib.colors import ListedColormap

X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                     stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                     stop = X_set[:, 1].max() + 1, step = 0.01))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)

plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend() # to show legend

# show scatter plot
plt.show()
```

```
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA se
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA se
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA se
```



```

"""plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
             X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
             cmap = ListedColormap(('yellow', 'white', 'aquamarine')))"""
```

Colab paid products - Cancel contracts here

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

```

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import plot_tree
from sklearn.metrics import classification_report
import graphviz
```

```python
train=pd.read_csv("/content/iris.csv")
train.head()
```

	sepallength	sepalwidth	petallength	petalwidth	class1
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```python
train.describe()
```

	sepallength	sepalwidth	petallength	petalwidth
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```python
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
```

```

# Encode labels in column 'species'.
train['class1']= label_encoder.fit_transform(train['class1'])

train['class1'].unique()
train.head()
```

	sepallength	sepalwidth	petallength	petalwidth	class1
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
a = np.array(train)
X=train.drop(columns=['class1'])
Y=train['class1']
```

```
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.9, random_state =1)
model = SVC(kernel='linear')
model=model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Predicted values:")
print(y_pred)
print("Confusion Matrix: ",confusion_matrix(y_test, y_pred))
print("Accuracy : ",accuracy_score(y_test,y_pred)*100)
print("Report : ",classification_report(y_test, y_pred))
# fitting x samples and y classes
model.fit(X, Y)

model.predict([[1,1,1,1]])

```

```
Predicted values:
[0 1 1 0 2 1 2 0 0 2 1 0 2 1 1 0 1 1 0 0 1 1 1 0 2 1 0 0 1 2 1 2 1 2 2 0 1
 0 1 2 2 0 1 2 1 2 0 0 0 1 0 0 2 2 2 2 1 1 2 1 0 2 1 0 0 2 0 2 2 1 1 2 2 0
 1 1 2 1 2 1 0 0 0 2 0 2 2 2 0 0 1 0 2 1 2 2 1 2 2 1 0 1 0 1 1 0 1 0 0 2 2
 2 0 0 2 0 2 0 2 1 0 2 0 1 0 1 1 0 0 1 0 1 1 0 1]
Confusion Matrix:  [[47  0  0]
 [ 0 42  2]
 [ 0  4 40]]
Accuracy :  95.55555555555556
Report :                precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       0.91      0.95      0.93        44
           2       0.95      0.91      0.93        44

    accuracy                           0.96       135
   macro avg       0.96      0.95      0.95       135
weighted avg       0.96      0.96      0.96       135
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not
  "X does not have valid feature names, but"
array([0])
```

```
model.predict([[5,6,7,8]])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not
  "X does not have valid feature names, but"
array([2])
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
#from sklearn.cluster import KMeans
#from yellowbrick.cluster import KElbowVisualizer
#from sklearn import metrics
```

```
df = pd.read_csv('/content/Live.csv')
df.drop(['status_id', 'status_published','Column1','Column2','Column3','Column4'], axis=1, i
df.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_w
0	video	529	512	262	432	92	
1	photo	150	0	0	150	0	
2	video	227	236	57	204	21	
3	photo	111	0	0	111	0	
4	photo	213	0	0	204	9	

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['status_type'] = le.fit_transform(df['status_type'])
df.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_w
0	3	529	512	262	432	92	
1	1	150	0	0	150	0	
2	3	227	236	57	204	21	
3	1	111	0	0	111	0	
4	1	213	0	0	204	9	

```
df.dropna(inplace=True)
df.head()
```

```
status_type  num_reactions  num_comments  num_shares  num_likes  num_loves  num_w
```

```
X = df
y = df['status_type']
```

```
1    1    150    0    0    150    0
```

```
y
```

```
0       3
1       1
2       3
3       1
4       1
       ..
7045    1
7046    1
7047    1
7048    1
7049    1
Name: status_type, Length: 7050, dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()
X = ms.fit_transform(X)
```

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
```

```
KMeans(n_clusters=2, random_state=0)
```

```
kmeans.cluster_centers_
```

```
array([[3.28506857e-01, 3.90710874e-02, 7.54854864e-04, 7.53667113e-04,
        3.85438884e-02, 2.17448568e-03, 2.43721364e-03, 1.20039760e-03,
        2.75348016e-03, 1.45313276e-03],
       [9.54921576e-01, 6.46330441e-02, 2.67028654e-02, 2.93171709e-02,
        5.71231462e-02, 4.71007076e-02, 8.18581889e-03, 9.65207685e-03,
        8.04219428e-03, 7.19501847e-03]])
```

```
kmeans.inertia_
```

```
237.75726404419646
```

```
labels = kmeans.labels_
# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
```

```
Result: 63 out of 7050 samples were correctly labeled.
```

```
import warnings
warnings.filterwarnings('ignore')

# Importing all required packages
import numpy as np
import pandas as pd

# Data viz lib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from matplotlib.pyplot import xticks
```

```
bank = pd.read_csv('/content/bankmarketing.csv')
```

```
bank.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	

5 rows × 21 columns

```
bank_cust = bank[['age','job', 'marital', 'education', 'default', 'housing', 'loan','contact
bank_cust.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	

```
bank_cust['age_bin'] = pd.cut(bank_cust['age'], [0, 20, 30, 40, 50, 60, 70, 80, 90, 100],
                              labels=['0-20', '20-30', '30-40', '40-50','50-60','60-70','70-
bank_cust  = bank_cust.drop('age',axis = 1)
```

```
bank_cust.head()
```

	job	marital	education	default	housing	loan	contact	month	day_of_we
0	housemaid	married	basic.4y	no	no	no	telephone	may	m
1	services	married	high.school	unknown	no	no	telephone	may	m
2	services	married	high.school	no	yes	no	telephone	may	m
3	admin.	married	basic.6y	no	no	no	telephone	may	m
4	services	married	high.school	no	no	yes	telephone	may	m

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
bank_cust = bank_cust.apply(le.fit_transform)
bank_cust.head()
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	pout
0	3	1	0	0	0	0	1	6	1	
1	7	1	3	1	0	0	1	6	1	
2	7	1	3	0	2	0	1	6	1	
3	0	1	1	0	0	0	1	6	1	
4	7	1	3	0	0	2	1	6	1	

```
bank_cust_copy = bank_cust.copy()
```

```
pip install kmodes
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting kmodes
  Downloading kmodes-0.12.2-py2.py3-none-any.whl (20 kB)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scikit-learn>=0.22.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: scipy>=0.13.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Installing collected packages: kmodes
Successfully installed kmodes-0.12.2
```

```
from kmodes.kmodes import KModes
```

```
km_cao = KModes(n_clusters=2, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(bank_cust)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 5322, cost: 192203.0
Run 1, iteration: 2/100, moves: 1160, cost: 192203.0
```

```
fitClusters_cao
```

```
array([1, 1, 0, ..., 0, 1, 0], dtype=uint16)
```

```
clusterCentroidsDf = pd.DataFrame(km_cao.cluster_centroids_)
clusterCentroidsDf.columns = bank_cust.columns
```

```
clusterCentroidsDf
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	pout
0	0	1	6	0	2	0	0	6	2	
1	1	1	3	0	0	0	1	6	0	

```
km_huang = KModes(n_clusters=2, init = "Huang", n_init = 1, verbose=1)
fitClusters_huang = km_huang.fit_predict(bank_cust)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 3724, cost: 195568.0
```

```
fitClusters_huang
```

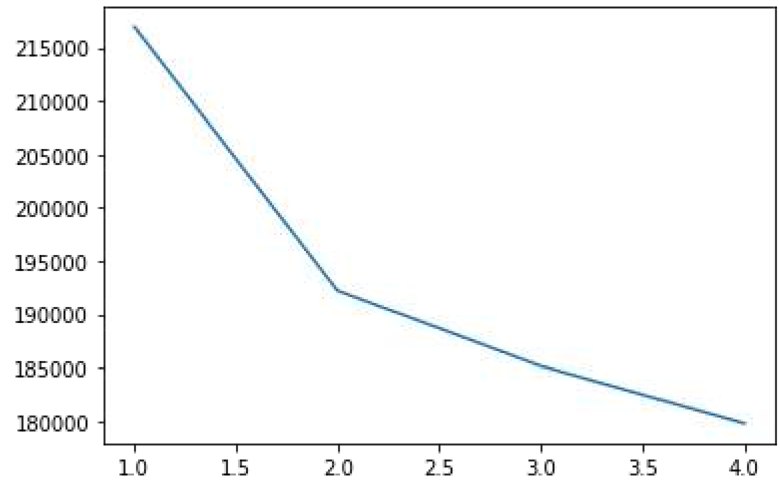
```
array([1, 1, 1, ..., 0, 0, 0], dtype=uint16)
```

```
cost = []
for num_clusters in list(range(1,5)):
    kmode = KModes(n_clusters=num_clusters, init = "Cao", n_init = 1, verbose=1)
    kmode.fit_predict(bank_cust)
    cost.append(kmode.cost_)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 216952.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 5322, cost: 192203.0
Run 1, iteration: 2/100, moves: 1160, cost: 192203.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 4993, cost: 185138.0
Run 1, iteration: 2/100, moves: 1368, cost: 185138.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 6186, cost: 179774.0
Run 1, iteration: 2/100, moves: 1395, cost: 179774.0
```

```
y = np.array([i for i in range(1,5,1)])
plt.plot(y,cost)
```

```
[<matplotlib.lines.Line2D at 0x7fca833f53d0>]
```



```
km_cao = KModes(n_clusters=2, init = "Cao", n_init = 1, verbose=1)
fitClusters_cao = km_cao.fit_predict(bank_cust)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 5322, cost: 192203.0
Run 1, iteration: 2/100, moves: 1160, cost: 192203.0
```

```
fitClusters_cao
```

```
array([1, 1, 0, ..., 0, 1, 0], dtype=uint16)
```

```
bank_cust = bank_cust_copy.reset_index()
```

```
clustersDf = pd.DataFrame(fitClusters_cao)
clustersDf.columns = ['cluster_predicted']
combinedDf = pd.concat([bank_cust, clustersDf], axis = 1).reset_index()
combinedDf = combinedDf.drop(['index', 'level_0'], axis = 1)
```

```
combinedDf.head()
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	pout
0	3	1	0	0	0	0	1	6	1	
1	7	1	3	1	0	0	1	6	1	
2	7	1	3	0	2	0	1	6	1	
3	0	1	1	0	0	0	1	6	1	
4	7	1	3	0	0	2	1	6	1	

```
import numpy as np
import pandas as pd
```

Double-click (or enter) to edit

```
data=pd.read_csv('/content/Iris.csv')
data.columns=['Id','Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Species']
data.head(10)
```

	Id	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

```
def activation_func(value):    #Tangent Hypotenuse
    #return (1/(1+np.exp(-value)))
    return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
```

```
def perceptron_train(in_data,labels,alpha):
    X=np.array(in_data)
    y=np.array(labels)
    weights=np.random.random(X.shape[1])
    original=weights
    bias=np.random.random_sample()
    for key in range(X.shape[0]):
        a=activation_func(np.matmul(np.transpose(weights),X[key]))
        yn=0
        if a>=0.7:
            yn=1
        elif a<=(-0.7):
            yn=-1
        weights=weights+alpha*(yn-y[key])*X[key]
        print('Iteration '+str(key)+': '+str(weights))
    print('Difference: '+str(weights-original))
    return weights
```

```
def perceptron_test(in_data,label_shape,weights):
    X=np.array(in_data)
    y=np.zeros(label_shape)
    for key in range(X.shape[1]):
        a=activation_func((weights*X[key]).sum())
        y[key]=0
        if a>=0.7:
            y[key]=1
        elif a<=(-0.7):
            y[key]=-1
    return y
```

```
def score(result,labels):
    difference=result-np.array(labels)
    correct_ctr=0
    for elem in range(difference.shape[0]):
        if difference[elem]==0:
            correct_ctr+=1
    score=correct_ctr*100/difference.size
    print('Score='+str(score))
```

```
# Dividing DataFrame "data" into "d_train" (60%) and "d_test" (40%)
divider = np.random.rand(len(data)) < 0.70
d_train=data[divider]
d_test=data[~divider]
```

```
# Dividing d_train into data and labels/targets
d_train_y=d_train['Species']
d_train_X=d_train.drop(['Species'],axis=1)

# Dividing d_train into data and labels/targets
d_test_y=d_test['Species']
d_test_X=d_test.drop(['Species'],axis=1)
```

```
# Learning rate
alpha = 0.01

# Train
weights = perceptron_train(d_train_X, d_train_y, alpha)
```

```
# Test
result_test=perceptron_test(d_test_X,d_test_y.shape,weights)
```

```
# Calculate score
score(result_test,d_test_y)
```

```
Score=28.571428571428573
```

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
```

```
import matplotlib.pyplot as plt
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

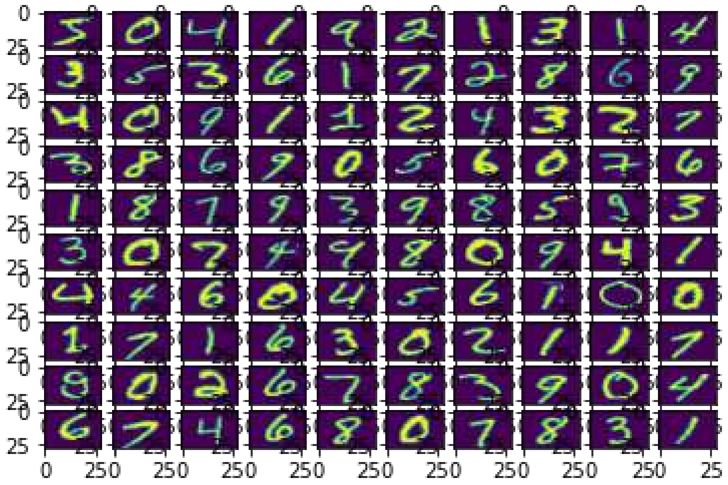
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
11490434/11490434 [==============================] - 0s 0us/step
```

```
# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# normalize image pixel values by dividing
# by 255
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale

```

```
print("Feature matrix:", x_train.shape)
print("Target matrix:", x_test.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_test.shape)

```

```
Feature matrix: (60000, 28, 28)
Target matrix: (10000, 28, 28)
Feature matrix: (60000,)
Target matrix: (10000,)
```

```
fig, ax = plt.subplots(10, 10)
k = 0
for i in range(10):
    for j in range(10):
        ax[i][j].imshow(x_train[k].reshape(28, 28),
                        aspect='auto')
        k += 1
plt.show()

```



```
model = Sequential([

    # reshape 28 row * 28 column data to 28*28 rows
    Flatten(input_shape=(28, 28)),

    # dense layer 1
    Dense(256, activation='sigmoid'),

    # dense layer 2
    Dense(128, activation='sigmoid'),

    # output layer
    Dense(10, activation='sigmoid'),
])

```

```
model.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

```

```
model.fit(x_train, y_train, epochs=10,
        batch_size=2000,
        validation_split=0.2)

```

```
Epoch 1/10
24/24 [==============================] - 2s 57ms/step - loss: 2.1259 - accuracy: 0.35
Epoch 2/10
24/24 [==============================] - 1s 48ms/step - loss: 1.4742 - accuracy: 0.72
Epoch 3/10
24/24 [==============================] - 1s 50ms/step - loss: 0.9484 - accuracy: 0.81
Epoch 4/10
24/24 [==============================] - 1s 50ms/step - loss: 0.6579 - accuracy: 0.85
Epoch 5/10
24/24 [==============================] - 1s 48ms/step - loss: 0.5093 - accuracy: 0.87
Epoch 6/10
24/24 [==============================] - 1s 48ms/step - loss: 0.4255 - accuracy: 0.89
Epoch 7/10
24/24 [==============================] - 1s 48ms/step - loss: 0.3738 - accuracy: 0.90
Epoch 8/10
24/24 [==============================] - 1s 48ms/step - loss: 0.3379 - accuracy: 0.90
Epoch 9/10
24/24 [==============================] - 1s 48ms/step - loss: 0.3118 - accuracy: 0.91
Epoch 10/10
24/24 [==============================] - 1s 49ms/step - loss: 0.2907 - accuracy: 0.92
<keras.callbacks.History at 0x7fa0595f94d0>
```

```
results = model.evaluate(x_test, y_test, verbose = 0)
print('test loss, test acc:', results)
```

```
test loss, test acc: [0.2749628722667694, 0.9228000044822693]
```

```
import numpy as np
import pandas as pd
```

```
data=pd.read_csv('/content/Iris.csv')
data.columns=['Id','Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Species']
data.head(10)
```

	Id	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
X, y = make_classification(n_samples=100, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,random_state=1)
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
clf.predict_proba(X_test[:1])
clf.predict(X_test[:5, :])
clf.score(X_test, y_test)
```

```
0.88
```

```
clf
```

```
MLPClassifier(max_iter=300, random_state=1)
```

```
import tkinter as tk
from tkinter import *
import cv2
from PIL import Image, ImageTk
import os
import numpy as np


```

```
global last_frame1
last_frame1 = np.zeros((480, 640, 3), dtype=np.uint8)
global last_frame2
last_frame2 = np.zeros((480, 640, 3), dtype=np.uint8)
global cap1
global cap2
cap1 = cv2.VideoCapture(-1)
cap2 = cv2.VideoCapture(-1)

def show_vid():
    if not cap1.isOpened():
        print("cant open the camera1")
    flag1, frame1 = cap1.read()
    frame1 = cv2.resize(frame1,(100,100))
    if flag1 is None:
        print ("Major error!")
    elif flag1:
        global last_frame1
        last_frame1 = frame1.copy()
        pic = cv2.cvtColor(last_frame1, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(pic)
        imgtk = ImageTk.PhotoImage(image=img)
        lmain.imgtk = imgtk
        lmain.configure(image=imgtk)
        lmain.after(10, show_vid)


if __name__ == '__main__':
    root=tk.Tk()
    lmain = tk.Label(master=root)
    lmain2 = tk.Label(master=root)

    lmain.pack(side = LEFT)
    lmain2.pack(side = RIGHT)
    root.title("Lane-line detection")
    root.geometry("900x700+100+10")
    exitbutton = Button(root, text='Quit',fg="red",command=   root.destroy).pack(side = BOTTOM
    show_vid()
    root.mainloop()
    cap.release()
```

```
---------------------------------------------------------------------------
TclError                                  Traceback (most recent call last)
<ipython-input-1-2d2e8826dcdc> in <module>
     35 
     36 if __name__ == '__main__':
---> 37     root=tk.Tk()
     38     lmain = tk.Label(master=root)
     39     lmain2 = tk.Label(master=root)

/usr/lib/python3.7/tkinter/__init__.py in __init__(self, screenName, baseName, className, useTk, sync, use)
   2021                 baseName = baseName + ext
   2022         interactive = 0
-> 2023         self.tk = _tkinter.create(screenName, baseName, className, interactive, wantobjects, useTk, sync, use)
   2024         if useTk:
   2025             self._loadtk()

TclError: no display name and no $DISPLAY environment variable
```

SEARCH STACK OVERFLOW

```
import numpy as np
import pandas as pd
```

+ Code + Text

```
data=pd.read_csv('/content/Iris.csv')
data.columns=['Id','Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Species']
data.head(10)
```

	Id	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
X, y = make_classification(n_samples=100, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,random_state=1)
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
clf.predict_proba(X_test[:1])
clf.predict(X_test[:5, :])
clf.score(X_test, y_test)
```

```
0.88
```

```
clf
```

```
MLPClassifier(max_iter=300, random_state=1)
```