In [ ]:

```
a= int(input("enter 1st no"))
b= int(input("enter 2nd no"))
c= int(input("enter 3rd no."))
if a>b and a>c :
    print(a,"is greatest")
elif b>a and b>c :
    print(b,"is greatest")
else:
    print(c," is greatest")
```

In [5]:

```
for i in range(1,21):
    if i%2==0:
        print(i,"is even")
    else:
        print(i, "is odd")
```

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
10 is even
11 is odd
12 is even
13 is odd
14 is even
15 is odd
16 is even
17 is odd
18 is even
19 is odd
20 is even
```

In [26]:

```
num = input("Enter any no = ")
sum=0
for i in (num):
    sum += int(i)
print("Sum of digits is ",sum)
```

```
Enter any no = 12345
Sum of digits is  15
```

In [30]:

```
num= input("enter no.")
last=len(num)-1
start=0
while(start<=last):
    if num[start]!=num[last]:
        print("no palindrom")
        last=0
        break
    start+=i
    last+-i
if last!=0:
    print(num, "a palindrome no.")
```

```
enter no.121

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\AMRESH~1\AppData\Local\Temp/ipykernel_5468/223682299.py in <module>
      7         last=0
      8         break
----> 9     start+=i
     10     last+-i
     11 if last!=0:

TypeError: unsupported operand type(s) for +=: 'int' and 'str'
```

In [32]:

```
Num = int(input("Enter a value:"))
Temp = num
Rev = 0
while(num>0):
    dig = num % 10
    revrev = rev * 10 + dig
    numnum = num // 10
if(temp == rev):
    print("This value is a palindrome number!")
else:
    print("This value is not a palindrome number!")
```

```
Enter a value:111

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\AMRESH~1\AppData\Local\Temp/ipykernel_5468/2920106691.py in <module>
      2 Temp = num
      3 Rev = 0
----> 4 while(num>0):
      5     dig = num % 10
      6     revrev = rev * 10 + dig

TypeError: '>' not supported between instances of 'str' and 'int'
```

In [33]:
```
import numpy as np
import pandas as pd
```

In [34]:
```
df=pd.read_csv('spambase.csv')
```

In [35]:
```
df
```

Out[35]:

	0	0.64	0.64.1	0.1	0.32	0.2	0.3	0.4	0.5	0.6	...	0.40	0.41	0.42	0.778	0.4
0	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.000	0.132	0.0	0.372	0.18
1	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.010	0.143	0.0	0.276	0.18
2	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.000	0.137	0.0	0.137	0.00
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.000	0.135	0.0	0.135	0.00
4	0.00	0.00	0.00	0.0	1.85	0.00	0.00	1.85	0.00	0.00	...	0.000	0.223	0.0	0.000	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
4595	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	...	0.000	0.232	0.0	0.000	0.00
4596	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.0	0.353	0.00
4597	0.30	0.00	0.30	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.102	0.718	0.0	0.000	0.00
4598	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.000	0.057	0.0	0.000	0.00
4599	0.00	0.00	0.65	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.0	0.125	0.00

4600 rows × 58 columns

In [36]:
```
print("Top 5 data in given ")
df.head(5)
```

```
Top 5 data in given
```

Out[36]:

	0	0.64	0.64.1	0.1	0.32	0.2	0.3	0.4	0.5	0.6	...	0.40	0.41	0.42	0.778	0.43	
0	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.00	0.132	0.0	0.372	0.180	0
1	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.01	0.143	0.0	0.276	0.184	0
2	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.137	0.0	0.137	0.000	0
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.135	0.0	0.135	0.000	0
4	0.00	0.00	0.00	0.0	1.85	0.00	0.00	1.85	0.00	0.00	...	0.00	0.223	0.0	0.000	0.000	0

5 rows × 58 columns

In [37]:
```
df.tail(5)
```

Out[37]:

	0	0.64	0.64.1	0.1	0.32	0.2	0.3	0.4	0.5	0.6	...	0.40	0.41	0.42	0.778	0.43	0.4
4595	0.31	0.0	0.62	0.0	0.00	0.31	0.0	0.0	0.0	0.0	...	0.000	0.232	0.0	0.000	0.0	0.
4596	0.00	0.0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.0	...	0.000	0.000	0.0	0.353	0.0	0.
4597	0.30	0.0	0.30	0.0	0.00	0.00	0.0	0.0	0.0	0.0	...	0.102	0.718	0.0	0.000	0.0	0.
4598	0.96	0.0	0.00	0.0	0.32	0.00	0.0	0.0	0.0	0.0	...	0.000	0.057	0.0	0.000	0.0	0.
4599	0.00	0.0	0.65	0.0	0.00	0.00	0.0	0.0	0.0	0.0	...	0.000	0.000	0.0	0.125	0.0	0.

5 rows × 58 columns

In [38]:
```
df.shape
```

Out[38]:
```
(4600, 58)
```

In [48]:
```
[row,col]=df.shape
```

In [49]:
```
Data = df.iloc[0 : row, 0: (col-1)]
Label = df.iloc[0: row, (col-1)]
```

In [50]:
```
Data.shape
```

Out[50]:
```
(4600, 57)
```

In [51]:
```
Label.value_counts()
```

Out[51]:
```
0    2788
1    1812
Name: 1, dtype: int64
```

```
In [52]: Data
```

```
Out[52]:
          0  0.64  0.64.1  0.1  0.32   0.2   0.3   0.4   0.5   0.6  ...  0.39   0.40   0.41  0.42  0.778
   0   0.21  0.28    0.50  0.0  0.14  0.28  0.21  0.07  0.00  0.94  ...   0.0  0.000  0.132   0.0  0.372
   1   0.06  0.00    0.71  0.0  1.23  0.19  0.19  0.12  0.64  0.25  ...   0.0  0.010  0.143   0.0  0.276
   2   0.00  0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...   0.0  0.000  0.137   0.0  0.137
   3   0.00  0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...   0.0  0.000  0.135   0.0  0.135
   4   0.00  0.00    0.00  0.0  1.85  0.00  0.00  1.85  0.00  0.00  ...   0.0  0.000  0.223   0.0  0.000
 ...    ...   ...     ...  ...   ...   ...   ...   ...   ...   ...  ...   ...    ...    ...   ...    ...
4595   0.31  0.00    0.62  0.0  0.00  0.31  0.00  0.00  0.00  0.00  ...   0.0  0.000  0.232   0.0  0.000
4596   0.00  0.00    0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...   0.0  0.000  0.000   0.0  0.353
4597   0.30  0.00    0.30  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...   0.0  0.102  0.718   0.0  0.000
4598   0.96  0.00    0.00  0.0  0.32  0.00  0.00  0.00  0.00  0.00  ...   0.0  0.000  0.057   0.0  0.000
4599   0.00  0.00    0.65  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...   0.0  0.000  0.000   0.0  0.125

4600 rows × 57 columns
```

```
In [53]: Data.describe()
```

```
Out[53]:
                 0         0.64       0.64.1          0.1         0.32          0.2          0.3
count  4600.000000  4600.000000  4600.000000  4600.000000  4600.000000  4600.000000  4600.000000
 mean     0.104576     0.212922     0.280578     0.065439     0.312222     0.095922     0.114233
  std     0.305387     1.290700     0.504170     1.395303     0.672586     0.273850     0.391480
  min     0.000000     0.000000     0.000000     0.000000     0.000000     0.000000     0.000000
  25%     0.000000     0.000000     0.000000     0.000000     0.000000     0.000000     0.000000
  50%     0.000000     0.000000     0.000000     0.000000     0.000000     0.000000     0.000000
  75%     0.000000     0.000000     0.420000     0.000000     0.382500     0.000000     0.000000
  max     4.540000    14.280000     5.100000    42.810000    10.000000     5.880000     7.270000

8 rows × 57 columns
```

In [56]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

x= Data
y= Label

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2)

knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(x_train, y_train)
print(knn.score(x_test, y_test))
```

```
0.7956521739130434
```

In [57]:

```
y_test.value_counts()
```

Out[57]:

```
0    563
1    357
Name: 1, dtype: int64
```

In [58]:

```
y_train.value_counts()
```

Out[58]:

```
0    2225
1    1455
Name: 1, dtype: int64
```

In [59]:

```
y_train.value_counts() + y_test.value_counts()
```

Out[59]:

```
0    2788
1    1812
Name: 1, dtype: int64
```

In [60]:

```
Label.count()
```

Out[60]:

```
4600
```

In [61]:

```
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

iris = datasets.load_iris()
X, y = iris.data[:, :], iris.target
Xtrain, Xtest, y_train, y_test = train_test_split(X, y, stratify = y, random_stat

scaler = preprocessing.StandardScaler().fit(Xtrain)
Xtrain = scaler.transform(Xtrain)
Xtest = scaler.transform(Xtest)

knn = neighbors.KNeighborsClassifier(n_neighbors=3)
knn.fit(Xtrain, y_train)
y_pred = knn.predict(Xtest)

print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
0.9777777777777777
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.94      1.00      0.97        15
           2       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

[[15  0  0]
 [ 0 15  0]
 [ 0  1 14]]
```

In [68]:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

In [72]:

```
n = int(input("enter no."))
```

```
enter no.5
```

In [73]:

```
print(factorial(n))
```

```
120
```

In [75]:
```
def UniqueList(UL):
    s = []
    for a in UL:
        if a not in s:
            s.append(a)
        return s
```

In [76]:
```
print(UniqueList([6,3,5,3,5,3,4,6,2,9,6,9,1,7,7,10,8]))
```

```
[6]
```

In [77]:
```
import numpy as np
A = np.array([5,10,15,20,25,30,35,40,45,50])
print("Entered Array List : \n", A)
ReversedArray = np.flip(A)
print("\nReversed Array List : \n", ReversedArray)
```

```
Entered Array List :
 [ 5 10 15 20 25 30 35 40 45 50]

Reversed Array List :
 [50 45 40 35 30 25 20 15 10  5]
```

In [78]:
```
import pandas as pd
dig = {'Dairymilk': [40, 50, 90, 100, 250],
'DarkChocolate': [300, 200, 100, 250, 500],
'Perk': [50, 60, 45, 80, 90],
'Fuse': [100, 150, 300, 200, 250],
'KitKat': [110, 200, 150, 60, 70]}
```

In [79]:
```
df = pd.DataFrame(data=dig)
```

In [80]:
```
df
```

Out[80]:

	Dairymilk	DarkChocolate	Perk	Fuse	KitKat
0	40	300	50	100	110
1	50	200	60	150	200
2	90	100	45	300	150
3	100	250	80	200	60
4	250	500	90	250	70

In [81]:
```
import pandas as pd
df = pd.read_csv("USA_Housing.csv")
```

In [82]:
```
df
```

Out[82]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Addres
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Ap 674\nLaurabury, N 3701.
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson View Suite 079\nLak Kathleen, CA.
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabet Stravenue\nDanieltowr WI 06482.
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO A 4482
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFP(AE 0938
...	...	...	...	...	...	...	.
4995	60567.94414	7.830362	6.137356	3.46	22837.36103	1.060194e+06	USNS Williams\nFP(AP 30153-765
4996	78491.27543	6.999135	6.576763	4.02	25616.11549	1.482618e+06	PSC 9258, Bo 8489\nAPO AA 4299 335
4997	63390.68689	7.250591	4.805081	2.13	33266.14549	1.030730e+06	4215 Tracy Garde Suite 076\nJoshualan(VA 01.
4998	68001.33124	5.534388	7.130144	5.44	42625.62016	1.198657e+06	USS Wallace\nFPO A 7331
4999	65510.58180	5.992305	6.792336	4.07	46501.28380	1.298950e+06	37778 George Ridge Apt. 509\nEast Holl NV 2.

5000 rows × 7 columns

In [83]:
```
dfFirstFifty = df.head(50)
```

In [84]:
```
dfFirstFifty
```

	Income	House Age	Number of Rooms	Number of Bedrooms	Population	Price	Address
43	70421.47649	6.907083	[illegible]	[illegible]	43183.92849	1.744932e+06	541\nDavidso 346:
44	62614.42062	5.499310	7.440505	6.32	26888.57956	1.153871e+06	43087 Jer Field\nWest Debc SD 49
45	66394.87159	7.069512	7.204640	3.18	39741.07751	1.499989e+06	71956 Jen Fall\nBrooketown, 67485-(
46	73946.85107	4.863154	7.537182	6.35	35261.12702	1.109588e+06	8034 Pierce Prairie S 727\nDevonfurt, NE
47	69144.74571	7.296224	5.928223	3.22	19030.61549	9.801773e+05	Unit 8108 5159\nDPO AP 04
48	77278.69703	6.238891	6.919204	2.13	21725.95429	1.323952e+06	08639 Ga Port\nAnthonybury 17
49	86754.19663	6.604440	6.252455	4.02	43017.44076	1.662495e+06	91863 Curtis Point\nl Richard, AK 99996-7

In [85]:
```
df_column = df[['Avg. Area House Age', 'Avg. Area Number of Bedrooms', 'Price']]
```

In [86]:
```
df_column
```

Out[86]:

	Avg. Area House Age	Avg. Area Number of Bedrooms	Price
0	5.682861	4.09	1.059034e+06
1	6.002900	3.09	1.505891e+06
2	5.865890	5.13	1.058988e+06
3	7.188236	3.26	1.260617e+06
4	5.040555	4.23	6.309435e+05
...	...	...	...
4995	7.830362	3.46	1.060194e+06
4996	6.999135	4.02	1.482618e+06
4997	7.250591	2.13	1.030730e+06
4998	5.534388	5.44	1.198657e+06
4999	5.992305	4.07	1.298950e+06

5000 rows × 3 columns

In [87]:
```
import pandas as pd
df = pd.read_csv("spambase.csv")
```

In [88]:
```
df
```

Out[88]:

	0	0.64	0.64.1	0.1	0.32	0.2	0.3	0.4	0.5	0.6	...	0.40	0.41	0.42	0.778	0.4
0	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.000	0.132	0.0	0.372	0.18
1	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.010	0.143	0.0	0.276	0.18
2	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.000	0.137	0.0	0.137	0.00
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.000	0.135	0.0	0.135	0.00
4	0.00	0.00	0.00	0.0	1.85	0.00	0.00	1.85	0.00	0.00	...	0.000	0.223	0.0	0.000	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
4595	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	...	0.000	0.232	0.0	0.000	0.00
4596	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.0	0.353	0.00
4597	0.30	0.00	0.30	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.102	0.718	0.0	0.000	0.00
4598	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.000	0.057	0.0	0.000	0.00
4599	0.00	0.00	0.65	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.0	0.125	0.00

4600 rows × 58 columns

In [89]:
```
[row, col] = df.shape
Data = df.iloc[0 : row, 0 : (col - 1)]
Label = df.iloc[0 : row, (col - 1)]
```

In [90]:
```
Data
```

Out[90]:

	0	0.64	0.64.1	0.1	0.32	0.2	0.3	0.4	0.5	0.6	...	0.39	0.40	0.41	0.42	0.778
0	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.0	0.000	0.132	0.0	0.372
1	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.0	0.010	0.143	0.0	0.276
2	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.000	0.137	0.0	0.137
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.000	0.135	0.0	0.135
4	0.00	0.00	0.00	0.0	1.85	0.00	0.00	1.85	0.00	0.00	...	0.0	0.000	0.223	0.0	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4595	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	...	0.0	0.000	0.232	0.0	0.000
4596	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.000	0.0	0.353
4597	0.30	0.00	0.30	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.102	0.718	0.0	0.000
4598	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.057	0.0	0.000
4599	0.00	0.00	0.65	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.000	0.0	0.125

4600 rows × 57 columns

In [91]:
```
Label
```

Out[91]:
```
0       1
1       1
2       1
3       1
4       1
       ..
4595    0
4596    0
4597    0
4598    0
4599    0
Name: 1, Length: 4600, dtype: int64
```

In [95]:
```
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
X=Data
y=Label
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20,random_sta
kmns=KMeans(n_clusters=2, random_state=0)
kmns.fit(X_train, y_train)
KMeans(n_clusters=2, random_state=0)
predictions = kmns.predict(X_test)
print(predictions)
```

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

```
In [97]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test,predictions)

Out[97]: 0.5956521739130435
```

```
In [98]: df = pd.read_csv("heart_failure_clinical_records_dataset.csv")
```

```
In [99]: [row,col]= df.shape
         Data = df.iloc[0:row, 0:(col -1)]
         Label = df.iloc[0:row,(col -1)]
```

```
In [101]: from sklearn.model_selection import train_test_split
          from sklearn.cluster import KMeans
          X=Data
          y=Label
          X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20,random_sta
          42)
          kmns=KMeans(n_clusters=2,random_state=0)
          kmns.fit(X_train,y_train)

Out[101]: KMeans(n_clusters=2, random_state=0)
```

```
In [102]: predictions = kmns.predict(X_test)
          print(predictions)

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

```
In [103]: from sklearn.metrics import accuracy_score
          accuracy_score(y_test, predictions)

Out[103]: 0.5833333333333334
```

```
In [104]: import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
```

```
In [105]: df = pd.read_csv('spambase.csv')
```

In [106]:
```
df
```

Out[106]:
```
         0   0.64  0.64.1  0.1  0.32   0.2   0.3   0.4   0.5   0.6  ...   0.40   0.41  0.42  0.778   0.4
   0  0.21   0.28    0.50  0.0  0.14  0.28  0.21  0.07  0.00  0.94  ...  0.000  0.132   0.0  0.372  0.18
   1  0.06   0.00    0.71  0.0  1.23  0.19  0.19  0.12  0.64  0.25  ...  0.010  0.143   0.0  0.276  0.18
   2  0.00   0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...  0.000  0.137   0.0  0.137  0.00
   3  0.00   0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...  0.000  0.135   0.0  0.135  0.00
   4  0.00   0.00    0.00  0.0  1.85  0.00  0.00  1.85  0.00  0.00  ...  0.000  0.223   0.0  0.000  0.00
 ...   ...    ...     ...  ...   ...   ...   ...   ...   ...   ...  ...    ...    ...   ...    ...     .
4595  0.31   0.00    0.62  0.0  0.00  0.31  0.00  0.00  0.00  0.00  ...  0.000  0.232   0.0  0.000  0.00
4596  0.00   0.00    0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...  0.000  0.000   0.0  0.353  0.00
4597  0.30   0.00    0.30  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...  0.102  0.718   0.0  0.000  0.00
4598  0.96   0.00    0.00  0.0  0.32  0.00  0.00  0.00  0.00  0.00  ...  0.000  0.057   0.0  0.000  0.00
4599  0.00   0.00    0.65  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...  0.000  0.000   0.0  0.125  0.00

4600 rows × 58 columns
```

In [107]:
```
[row,col]=df.shape
Data = df.iloc[0:row, 0: (col-1)]
Label=df.iloc[0:row, (col-1)]
```

In [108]:
```
# DT
```

In [110]:
```
ort warnings
nings.filterwarnings("ignore")
m sklearn.tree import DecisionTreeClassifier
m sklearn.model_selection import train_test_split
m sklearn.metrics import confusion_matrix
m sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
```

In [115]:
```
_train, X_test, y_train, y_test = train_test_split(Data, Label, test_size =0.1)
```

In [116]:
```
dt = DecisionTreeClassifier(random_state = 0, max_depth = 2)
dt.fit(X_train, y_train )
y_pred = dt.predict(X_test)
```

In [117]:
```
print('Accuracy: %.4f' %accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8370
```

In [118]:
```
# kNN
```

In [119]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
X = Data
y = Label
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
knn = KNeighborsClassifier(n_neighbors = 4)
knn.fit( X_train, y_train)
print(knn.score(X_test,y_test))
```

```
0.7978260869565217
```

In [120]:
```
# k-Means
```

In [124]:
```
learn.model_selection import train_test_split
learn.cluster import KMeans
a
el
,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.20, random_state=42)
Means(n_clusters = 2, random_state = 0)
t( X_train, y_train)
```

Out[124]:
```
KMeans(n_clusters=2, random_state=0)
```

In [125]:
```
import numpy as np
import pandas as pd
df=pd.read_csv('spambase.csv')
df
```

Out[125]:
```
         0  0.64  0.64.1  0.1  0.32   0.2   0.3   0.4   0.5   0.6  ...   0.40   0.41  0.42  0.778   0.4
   0  0.21  0.28    0.50  0.0  0.14  0.28  0.21  0.07  0.00  0.94  ...  0.000  0.132   0.0  0.372  0.18
   1  0.06  0.00    0.71  0.0  1.23  0.19  0.19  0.12  0.64  0.25  ...  0.010  0.143   0.0  0.276  0.18
   2  0.00  0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...  0.000  0.137   0.0  0.137  0.00
   3  0.00  0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...  0.000  0.135   0.0  0.135  0.00
   4  0.00  0.00    0.00  0.0  1.85  0.00  0.00  1.85  0.00  0.00  ...  0.000  0.223   0.0  0.000  0.00
 ...   ...   ...     ...  ...   ...   ...   ...   ...   ...   ...  ...    ...    ...   ...    ...     .
4595  0.31  0.00    0.62  0.0  0.00  0.31  0.00  0.00  0.00  0.00  ...  0.000  0.232   0.0  0.000  0.00
4596  0.00  0.00    0.00  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...  0.000  0.000   0.0  0.353  0.00
4597  0.30  0.00    0.30  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...  0.102  0.718   0.0  0.000  0.00
4598  0.96  0.00    0.00  0.0  0.32  0.00  0.00  0.00  0.00  0.00  ...  0.000  0.057   0.0  0.000  0.00
4599  0.00  0.00    0.65  0.0  0.00  0.00  0.00  0.00  0.00  0.00  ...  0.000  0.000   0.0  0.125  0.00

4600 rows × 58 columns
```

In [126]:
```
df.head()
```

Out[126]:
```
      0  0.64  0.64.1  0.1  0.32   0.2   0.3   0.4   0.5   0.6  ...  0.40   0.41  0.42  0.778   0.43
0  0.21  0.28    0.50  0.0  0.14  0.28  0.21  0.07  0.00  0.94  ...  0.00  0.132   0.0  0.372  0.180  0
1  0.06  0.00    0.71  0.0  1.23  0.19  0.19  0.12  0.64  0.25  ...  0.01  0.143   0.0  0.276  0.184  0
2  0.00  0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...  0.00  0.137   0.0  0.137  0.000  0
3  0.00  0.00    0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...  0.00  0.135   0.0  0.135  0.000  0
4  0.00  0.00    0.00  0.0  1.85  0.00  0.00  1.85  0.00  0.00  ...  0.00  0.223   0.0  0.000  0.000  0

5 rows × 58 columns
```

In [127]:

```
df.tail()
```

Out[127]:

	0	0.64	0.64.1	0.1	0.32	0.2	0.3	0.4	0.5	0.6	...	0.40	0.41	0.42	0.778	0.43	0.4
4595	0.31	0.0	0.62	0.0	0.00	0.31	0.0	0.0	0.0	0.0	...	0.000	0.232	0.0	0.000	0.0	0.
4596	0.00	0.0	0.00	0.0	0.00	0.00	0.0	0.0	0.0	0.0	...	0.000	0.000	0.0	0.353	0.0	0.
4597	0.30	0.0	0.30	0.0	0.00	0.00	0.0	0.0	0.0	0.0	...	0.102	0.718	0.0	0.000	0.0	0.
4598	0.96	0.0	0.00	0.0	0.32	0.00	0.0	0.0	0.0	0.0	...	0.000	0.057	0.0	0.000	0.0	0.
4599	0.00	0.0	0.65	0.0	0.00	0.00	0.0	0.0	0.0	0.0	...	0.000	0.000	0.0	0.125	0.0	0.

5 rows × 58 columns

In [128]:

```
df.shape
```

Out[128]:

```
(4600, 58)
```

In [129]:

```
[row, col] = df.shape
Data = df.iloc[0 : row, 0 : (col - 1)]
Label = df.iloc[0 : row, (col - 1)]
Data
```

Out[129]:

	0	0.64	0.64.1	0.1	0.32	0.2	0.3	0.4	0.5	0.6	...	0.39	0.40	0.41	0.42	0.778
0	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.0	0.000	0.132	0.0	0.372
1	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.0	0.010	0.143	0.0	0.276
2	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.000	0.137	0.0	0.137
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.0	0.000	0.135	0.0	0.135
4	0.00	0.00	0.00	0.0	1.85	0.00	0.00	1.85	0.00	0.00	...	0.0	0.000	0.223	0.0	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4595	0.31	0.00	0.62	0.0	0.00	0.31	0.00	0.00	0.00	0.00	...	0.0	0.000	0.232	0.0	0.000
4596	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.000	0.0	0.353
4597	0.30	0.00	0.30	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.102	0.718	0.0	0.000
4598	0.96	0.00	0.00	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.057	0.0	0.000
4599	0.00	0.00	0.65	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.000	0.0	0.125

4600 rows × 57 columns

In [130]:

```
Label
```

Out[130]:

```
0       1
1       1
2       1
3       1
4       1
       ..
4595    0
4596    0
4597    0
4598    0
4599    0
Name: 1, Length: 4600, dtype: int64
```

In [131]:

```
Label.value_counts()
```

Out[131]:

```
0    2788
1    1812
Name: 1, dtype: int64
```

In [132]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
X = Data
y = Label
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
knn = KNeighborsClassifier(n_neighbors = 4)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.8152173913043478
```

In [133]:

```
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.825
```

In [134]:

```
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.8152173913043478
```

In [135]:

```
knn = KNeighborsClassifier(n_neighbors = 10)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.7934782608695652
```

In [136]:

```
knn = KNeighborsClassifier(n_neighbors = 20)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.7771739130434783
```

In [137]:

```
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.825
```

In [138]:

```
#precision in knn
from sklearn.metrics import precision_score
y_pred=k_means.predict(X_test)
precision_score(y_test,y_pred,average=None,zero_division=1)[0]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\AMRESH~1\AppData\Local\Temp/ipykernel_5468/2936095963.py in <module>
      1 #precision in knn
      2 from sklearn.metrics import precision_score
----> 3 y_pred=k_means.predict(X_test)
      4 precision_score(y_test,y_pred,average=None,zero_division=1)[0]

NameError: name 'k_means' is not defined
```

In [139]:

```
#Recall_score in knn

from sklearn.metrics import recall_score
recall_score(y_test,y_pred,average='macro',zero_division=1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\AMRESH~1\AppData\Local\Temp/ipykernel_5468/2000363971.py in <module>
      2 
      3 from sklearn.metrics import recall_score
----> 4 recall_score(y_test,y_pred,average='macro',zero_division=1)

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64 
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in recall_score(y_true, y_pred, labels, pos_label, average, sample_weight, zero_division)
   1772     array([0.5, 1. , 1. ])
   1773     """
-> 1774     _, r, _, _ = precision_recall_fscore_support(y_true, y_pred,
   1775                                                  labels=labels,
   1776                                                  pos_label=pos_label,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64 
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in precision_recall_fscore_support(y_true, y_pred, beta, labels, pos_label, average, warn_for, sample_weight, zero_division)
   1462     if beta < 0:
   1463         raise ValueError("beta should be >=0 in the F-beta score")
-> 1464     labels = _check_set_wise_labels(y_true, y_pred, average, labels,
   1465                                     pos_label)
   1466 

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in _check_set_wise_labels(y_true, y_pred, average, labels, pos_label)
   1275                              str(average_options))
   1276 
-> 1277     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
   1278     # Convert to Python primitive type to avoid NumPy type / Python str
   1279     # comparison. See https://github.com/numpy/numpy/issues/6784 (https://github.com/numpy/numpy/issues/6784)

~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py in _check_ta
```

```
rgets(y_true, y_pred)
     81     y_pred : array or indicator matrix
     82     """
---> 83     check_consistent_length(y_true, y_pred)
     84     type_true = type_of_target(y_true)
     85     type_pred = type_of_target(y_pred)

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_consistent
_length(*arrays)
    317     uniques = np.unique(lengths)
    318     if len(uniques) > 1:
--> 319         raise ValueError("Found input variables with inconsistent num
bers of"
    320                          " samples: %r" % [int(l) for l in lengths])
    321 

ValueError: Found input variables with inconsistent numbers of samples: [920,
460]
```

```
In [ ]:
```