LAB 2

Navneeth Krishna 21047

Part A: Prerequisite for KNN implementation

1) Creating 2 vectors

In [585…

```
import numpy as np
V1 =np.array([9,8,7,6])
V2 = np.array([3,5,9,2])
```

Checking which values are same

In [586…

```
V3 = np.sum(V1 == V2)
print(V3)
```

```
0
```

2) Matrix creation

a)Creating a matrix with 10 rows and 3 columns

In [587…

```
M = np.random.randint(10,size=(10,3))
print(M)
```

```
[[3 8 8]
 [7 8 1]
 [6 4 5]
 [5 2 5]
 [9 7 3]
 [8 1 6]
 [8 1 0]
 [4 2 7]
 [1 9 9]
 [1 9 6]]
```

b.Printing the size of M

In [588…

```
M.shape
```

Out[588]:

```
(10, 3)
```

c.Printing the number of rows in M

In [589…

```
M.shape[0]
```

Out[589]:

```
10
```

d.Printing the columns in M

In [590…
```
M.shape[1]
```

Out[590]:
```
3
```

e.Simple loop to modify 3rd column

In [591…
```
C1 = np.row_stack (M[:,0])
C2 = np.row_stack (M[:,1])
C3 = np.row_stack (M[:,2])
arr1 = np.column_stack((C1,C2))
N = np.array(arr1)
Sum = C1 + C2
length = np.size(Sum)
for X in range(length):
  if np.mod(Sum[X],4) == 0:
    C3[X] = 1
  else:
    C3[X] = 0

arr2 = np.column_stack((arr1,C3))
M = arr2
print(M)
```

```
[[3 8 0]
 [7 8 0]
 [6 4 0]
 [5 2 0]
 [9 7 1]
 [8 1 0]
 [8 1 0]
 [4 2 0]
 [1 9 0]
 [1 9 0]]
```

3.Creating pandas data frame df

In [592…
```
import pandas as pd
df = pd.DataFrame(M)
Y = M[0:,0:,]
print(df)
```

```
   0  1  2
0  3  8  0
1  7  8  0
2  6  4  0
3  5  2  0
4  9  7  1
5  8  1  0
6  8  1  0
7  4  2  0
8  1  9  0
9  1  9  0
```

Naming the columns

In [593…

```
df.rename (columns = {  0  : 'X1' ,  1  : 'X2' ,  2  : 'Y' }, inplace=True )
df
```

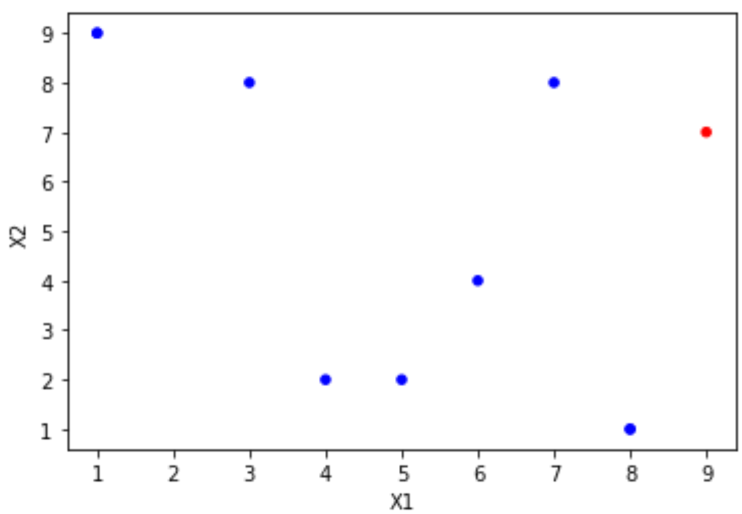
Out[593]:

	X1	X2	Y
0	3	8	0
1	7	8	0
2	6	4	0
3	5	2	0
4	9	7	1
5	8	1	0
6	8	1	0
7	4	2	0
8	1	9	0
9	1	9	0

4.Plot X1 and X2 using scatter plot

In [594…

```
import matplotlib.pyplot as plt
col = df.Y.map({0:'b', 1:'r'})
df.plot.scatter(x='X1', y='X2', c=col)
plt.show()
```



5.a.Find squared error

In [595…

```
SE = np.square(C1 - C2)
print(SE)
```

```
[[25]
 [ 1]
 [ 4]
 [ 9]
 [ 4]
 [49]
 [49]
 [ 4]
 [64]
 [64]]
```

b.Sum of squared error

In [596…

```
SSE = np.sum(SE)
SSE
```

Out[596]:

```
273
```

6.Find euclidian distance between first 2 rows

In [597…

```
import math as math
p = M[0,:]
q = M[1,:]
#ED = math.dist(p,q)
euclidiean_distance = np.sqrt(np.sum((p-q)**2))
print(euclidiean_distance)
```

```
4.0
```

Compare the euclidian distance

In [598…

```
comp = np.linalg.norm(p-q)
print(comp)
```

```
4.0
```

7.Create vector with random values

In [599…

```
import numpy as np
V = np.random.randint(10, size=(2))
print(V)
```

```
[1 3]
```

Finding euclidian distance between M and V and storing it and printing

In [600…

```
lis = list()
for i in M:
    lis.append(np.linalg.norm(i[:2]-V))
dis = np.array(lis)
print(dis)
```

```
[5.385 7.81  5.099 4.123 8.944 7.28  7.28  3.162 6.    6.   ]
```

8.Manipulate matrix

Create a matrix A with 10 rows and 2 columns.

In [601…
```
A=np.array([[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[7,8],[8,9],[9,1],[3,4]])
print(A)
```

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

Add new column

In [602…
```
#C = np.array[[4],[3],[2],[6],[8],[3],[0],[2],[7],[3]]
C = np.array([4,3,2,6,8,3,0,2,7,3])
A=np.column_stack((A,C))
print(A)
```

```
[[1 2 4]
 [2 3 3]
 [3 4 2]
 [4 5 6]
 [5 6 8]
 [6 7 3]
 [7 8 0]
 [8 9 2]
 [9 1 7]
 [3 4 3]]
```

Add new row to matrix

In [603…
```
R = np.array([[4,1,6]])
A = np.vstack((A,R))
print(A)
```

```
[[1 2 4]
 [2 3 3]
 [3 4 2]
 [4 5 6]
 [5 6 8]
 [6 7 3]
 [7 8 0]
 [8 9 2]
 [9 1 7]
 [3 4 3]
 [4 1 6]]
```

9.Create a matrix Md with two columns X1, X2 and populate with random values

In [604…
```
Md = np.random.randint(10,size=(10,2))
colname=['X1','X2']
df = pd.DataFrame(Md, columns=colname)
print(df)
```

```
   X1  X2
0   7   9
1   0   7
2   2   1
3   5   6
4   7   1
5   1   5
6   1   3
7   1   4
8   8   7
9   8   4
```

In [605…

```
z = np.random.randint(2,size = (10,1))
Md = np.column_stack((Md,z))
print(Md)
```

```
[[7 9 1]
 [0 7 0]
 [2 1 1]
 [5 6 0]
 [7 1 0]
 [1 5 0]
 [1 3 1]
 [1 4 0]
 [8 7 0]
 [8 4 0]]
```

Euclidean distance between Md and M

In [606…

```
m = np.random.randint(50, size=(10,2))
Dist=np.empty([100,3])
index=0
for i in range(m.shape[0]):
  for j in range(m.shape[0]):
    e_dis=np.sqrt(np.sum((M[i]-Md[j])**2))
    temp=[i , j , e_dis]
    Dist[index]=temp
    index+=1
print(Dist)
```

```
[[ 0.     0.     4.243]
 [ 0.     1.     3.162]
 [ 0.     2.     7.141]
 [ 0.     3.     2.828]
 [ 0.     4.     8.062]
 [ 0.     5.     3.606]
 [ 0.     6.     5.477]
 [ 0.     7.     4.472]
 [ 0.     8.     5.099]
 [ 0.     9.     6.403]
 [ 1.     0.     1.414]
 [ 1.     1.     7.071]
 [ 1.     2.     8.66 ]
 [ 1.     3.     2.828]
 [ 1.     4.     7.   ]
 [ 1.     5.     6.708]
 [ 1.     6.     7.874]
 [ 1.     7.     7.211]
 [ 1.     8.     1.414]
 [ 1.     9.     4.123]
 [ 2.     0.     5.196]
 [ 2.     1.     6.708]
 [ 2.     2.     5.099]
 [ 2.     3.     2.236]
 [ 2.     4.     3.162]
 [ 2.     5.     5.099]
 [ 2.     6.     5.196]
 [ 2.     7.     5.   ]
 [ 2.     8.     3.606]
 [ 2.     9.     2.   ]
 [ 3.     0.     7.348]
 [ 3.     1.     7.071]
 [ 3.     2.     3.317]
 [ 3.     3.     4.   ]
 [ 3.     4.     2.236]
 [ 3.     5.     5.   ]
 [ 3.     6.     4.243]
 [ 3.     7.     4.472]
 [ 3.     8.     5.831]
 [ 3.     9.     3.606]
 [ 4.     0.     2.828]
 [ 4.     1.     9.055]
 [ 4.     2.     9.22 ]
 [ 4.     3.     4.243]
 [ 4.     4.     6.403]
 [ 4.     5.     8.307]
 [ 4.     6.     8.944]
 [ 4.     7.     8.602]
 [ 4.     8.     1.414]
 [ 4.     9.     3.317]
 [ 5.     0.     8.124]
 [ 5.     1.    10.   ]
 [ 5.     2.     6.083]
 [ 5.     3.     5.831]
 [ 5.     4.     1.   ]
 [ 5.     5.     8.062]
 [ 5.     6.     7.348]
 [ 5.     7.     7.616]
 [ 5.     8.     6.   ]
```

```
 [ 5.     9.     3.   ]
 [ 6.     0.     8.124]
 [ 6.     1.    10.   ]
 [ 6.     2.     6.083]
 [ 6.     3.     5.831]
 [ 6.     4.     1.   ]
 [ 6.     5.     8.062]
 [ 6.     6.     7.348]
 [ 6.     7.     7.616]
 [ 6.     8.     6.   ]
 [ 6.     9.     3.   ]
 [ 7.     0.     7.681]
 [ 7.     1.     6.403]
 [ 7.     2.     2.449]
 [ 7.     3.     4.123]
 [ 7.     4.     3.162]
 [ 7.     5.     4.243]
 [ 7.     6.     3.317]
 [ 7.     7.     3.606]
 [ 7.     8.     6.403]
 [ 7.     9.     4.472]
 [ 8.     0.     6.083]
 [ 8.     1.     2.236]
 [ 8.     2.     8.124]
 [ 8.     3.     5.   ]
 [ 8.     4.    10.   ]
 [ 8.     5.     4.   ]
 [ 8.     6.     6.083]
 [ 8.     7.     5.   ]
 [ 8.     8.     7.28 ]
 [ 8.     9.     8.602]
 [ 9.     0.     6.083]
 [ 9.     1.     2.236]
 [ 9.     2.     8.124]
 [ 9.     3.     5.   ]
 [ 9.     4.    10.   ]
 [ 9.     5.     4.   ]
 [ 9.     6.     6.083]
 [ 9.     7.     5.   ]
 [ 9.     8.     7.28 ]
 [ 9.     9.     8.602]]
```

10.Sort Dist matrix based on last column.Use(print(a[a[:,n].argsort()])) where a is the matrix and n is the column based on which you need to sort.

In [607…

```
(print(Dist[Dist[:,1].argsort()]))
```

```
[[ 0.     0.     4.243]
 [ 8.     0.     6.083]
 [ 7.     0.     7.681]
 [ 6.     0.     8.124]
 [ 5.     0.     8.124]
 [ 4.     0.     2.828]
 [ 3.     0.     7.348]
 [ 2.     0.     5.196]
 [ 1.     0.     1.414]
 [ 9.     0.     6.083]
 [ 9.     1.     2.236]
 [ 1.     1.     7.071]
 [ 4.     1.     9.055]
 [ 0.     1.     3.162]
 [ 5.     1.    10.   ]
 [ 7.     1.     6.403]
 [ 3.     1.     7.071]
 [ 8.     1.     2.236]
 [ 2.     1.     6.708]
 [ 6.     1.    10.   ]
 [ 5.     2.     6.083]
 [ 3.     2.     3.317]
 [ 8.     2.     8.124]
 [ 4.     2.     9.22 ]
 [ 6.     2.     6.083]
 [ 1.     2.     8.66 ]
 [ 2.     2.     5.099]
 [ 9.     2.     8.124]
 [ 0.     2.     7.141]
 [ 7.     2.     2.449]
 [ 6.     3.     5.831]
 [ 1.     3.     2.828]
 [ 8.     3.     5.   ]
 [ 7.     3.     4.123]
 [ 5.     3.     5.831]
 [ 4.     3.     4.243]
 [ 3.     3.     4.   ]
 [ 9.     3.     5.   ]
 [ 0.     3.     2.828]
 [ 2.     3.     2.236]
 [ 7.     4.     3.162]
 [ 0.     4.     8.062]
 [ 6.     4.     1.   ]
 [ 5.     4.     1.   ]
 [ 4.     4.     6.403]
 [ 8.     4.    10.   ]
 [ 1.     4.     7.   ]
 [ 9.     4.    10.   ]
 [ 2.     4.     3.162]
 [ 3.     4.     2.236]
 [ 7.     5.     4.243]
 [ 1.     5.     6.708]
 [ 6.     5.     8.062]
 [ 2.     5.     5.099]
 [ 0.     5.     3.606]
 [ 9.     5.     4.   ]
 [ 4.     5.     8.307]
 [ 5.     5.     8.062]
 [ 3.     5.     5.   ]
```

```
 [ 8.     5.     4.   ]
 [ 2.     6.     5.196]
 [ 0.     6.     5.477]
 [ 8.     6.     6.083]
 [ 3.     6.     4.243]
 [ 7.     6.     3.317]
 [ 6.     6.     7.348]
 [ 4.     6.     8.944]
 [ 5.     6.     7.348]
 [ 9.     6.     6.083]
 [ 1.     6.     7.874]
 [ 7.     7.     3.606]
 [ 4.     7.     8.602]
 [ 9.     7.     5.   ]
 [ 5.     7.     7.616]
 [ 8.     7.     5.   ]
 [ 6.     7.     7.616]
 [ 3.     7.     4.472]
 [ 2.     7.     5.   ]
 [ 1.     7.     7.211]
 [ 0.     7.     4.472]
 [ 0.     8.     5.099]
 [ 6.     8.     6.   ]
 [ 5.     8.     6.   ]
 [ 4.     8.     1.414]
 [ 8.     8.     7.28 ]
 [ 3.     8.     5.831]
 [ 2.     8.     3.606]
 [ 1.     8.     1.414]
 [ 7.     8.     6.403]
 [ 9.     8.     7.28 ]
 [ 4.     9.     3.317]
 [ 7.     9.     4.472]
 [ 6.     9.     3.   ]
 [ 5.     9.     3.   ]
 [ 3.     9.     3.606]
 [ 2.     9.     2.   ]
 [ 1.     9.     4.123]
 [ 0.     9.     6.403]
 [ 8.     9.     8.602]
 [ 9.     9.     8.602]]
```

11.Get initial k rows from sorted matrix

In [608…
```
K = 10
for i in range(K):
  print(Md[i,:])
```

```
[7 9 1]
[0 7 0]
[2 1 1]
[5 6 0]
[7 1 0]
[1 5 0]
[1 3 1]
[1 4 0]
[8 7 0]
[8 4 0]
```

12.Find the number of 1s and number of 0s in k rows found above. Print 1 if the number of 1s are more else print 0.

In [609…
```
one = 0
zero = 0
for i in range(K):
  for j in range(3):
    if (Md[i][j] == 0):
      zero = zero + 1
    elif (Md[i][j] == 1):
      one = one + 1
if(one > zero):
    print(1)
else:
    print(0)
```

```
0
```

Part B: KNN implementation

a.Loading diabetes dataset

In [610…
```
from pandas import read_csv
data = read_csv('diabetes.csv')
```

b.Peek few columns

In [611…
```
print(data.head(5))
print(data.shape)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
(768, 9)
```

c.Splitting dataset to 80% training adn 20% testing using numpy slicing

In [612…

```
from sklearn.model_selection import train_test_split
test,training =data.values[:80,:], data.values[80:,:]
print(test)
```

```
[[6.000e+00 1.480e+02 7.200e+01 3.500e+01 0.000e+00 3.360e+01 6.270e-01
  5.000e+01 1.000e+00]
 [1.000e+00 8.500e+01 6.600e+01 2.900e+01 0.000e+00 2.660e+01 3.510e-01
  3.100e+01 0.000e+00]
 [8.000e+00 1.830e+02 6.400e+01 0.000e+00 0.000e+00 2.330e+01 6.720e-01
  3.200e+01 1.000e+00]
 [1.000e+00 8.900e+01 6.600e+01 2.300e+01 9.400e+01 2.810e+01 1.670e-01
  2.100e+01 0.000e+00]
 [0.000e+00 1.370e+02 4.000e+01 3.500e+01 1.680e+02 4.310e+01 2.288e+00
  3.300e+01 1.000e+00]
 [5.000e+00 1.160e+02 7.400e+01 0.000e+00 0.000e+00 2.560e+01 2.010e-01
  3.000e+01 0.000e+00]
 [3.000e+00 7.800e+01 5.000e+01 3.200e+01 8.800e+01 3.100e+01 2.480e-01
  2.600e+01 1.000e+00]
 [1.000e+01 1.150e+02 0.000e+00 0.000e+00 0.000e+00 3.530e+01 1.340e-01
  2.900e+01 0.000e+00]
 [2.000e+00 1.970e+02 7.000e+01 4.500e+01 5.430e+02 3.050e+01 1.580e-01
  5.300e+01 1.000e+00]
 [8.000e+00 1.250e+02 9.600e+01 0.000e+00 0.000e+00 0.000e+00 2.320e-01
  5.400e+01 1.000e+00]
 [4.000e+00 1.100e+02 9.200e+01 0.000e+00 0.000e+00 3.760e+01 1.910e-01
  3.000e+01 0.000e+00]
 [1.000e+01 1.680e+02 7.400e+01 0.000e+00 0.000e+00 3.800e+01 5.370e-01
  3.400e+01 1.000e+00]
 [1.000e+01 1.390e+02 8.000e+01 0.000e+00 0.000e+00 2.710e+01 1.441e+00
  5.700e+01 0.000e+00]
 [1.000e+00 1.890e+02 6.000e+01 2.300e+01 8.460e+02 3.010e+01 3.980e-01
  5.900e+01 1.000e+00]
 [5.000e+00 1.660e+02 7.200e+01 1.900e+01 1.750e+02 2.580e+01 5.870e-01
  5.100e+01 1.000e+00]
 [7.000e+00 1.000e+02 0.000e+00 0.000e+00 0.000e+00 3.000e+01 4.840e-01
  3.200e+01 1.000e+00]
 [0.000e+00 1.180e+02 8.400e+01 4.700e+01 2.300e+02 4.580e+01 5.510e-01
  3.100e+01 1.000e+00]
 [7.000e+00 1.070e+02 7.400e+01 0.000e+00 0.000e+00 2.960e+01 2.540e-01
  3.100e+01 1.000e+00]
 [1.000e+00 1.030e+02 3.000e+01 3.800e+01 8.300e+01 4.330e+01 1.830e-01
  3.300e+01 0.000e+00]
 [1.000e+00 1.150e+02 7.000e+01 3.000e+01 9.600e+01 3.460e+01 5.290e-01
  3.200e+01 1.000e+00]
 [3.000e+00 1.260e+02 8.800e+01 4.100e+01 2.350e+02 3.930e+01 7.040e-01
  2.700e+01 0.000e+00]
 [8.000e+00 9.900e+01 8.400e+01 0.000e+00 0.000e+00 3.540e+01 3.880e-01
  5.000e+01 0.000e+00]
 [7.000e+00 1.960e+02 9.000e+01 0.000e+00 0.000e+00 3.980e+01 4.510e-01
  4.100e+01 1.000e+00]
 [9.000e+00 1.190e+02 8.000e+01 3.500e+01 0.000e+00 2.900e+01 2.630e-01
  2.900e+01 1.000e+00]
 [1.100e+01 1.430e+02 9.400e+01 3.300e+01 1.460e+02 3.660e+01 2.540e-01
  5.100e+01 1.000e+00]
 [1.000e+01 1.250e+02 7.000e+01 2.600e+01 1.150e+02 3.110e+01 2.050e-01
  4.100e+01 1.000e+00]
 [7.000e+00 1.470e+02 7.600e+01 0.000e+00 0.000e+00 3.940e+01 2.570e-01
  4.300e+01 1.000e+00]
 [1.000e+00 9.700e+01 6.600e+01 1.500e+01 1.400e+02 2.320e+01 4.870e-01
  2.200e+01 0.000e+00]
 [1.300e+01 1.450e+02 8.200e+01 1.900e+01 1.100e+02 2.220e+01 2.450e-01
  5.700e+01 0.000e+00]
 [5.000e+00 1.170e+02 9.200e+01 0.000e+00 0.000e+00 3.410e+01 3.370e-01
```

```
 3.800e+01 0.000e+00]
[5.000e+00 1.090e+02 7.500e+01 2.600e+01 0.000e+00 3.600e+01 5.460e-01
 6.000e+01 0.000e+00]
[3.000e+00 1.580e+02 7.600e+01 3.600e+01 2.450e+02 3.160e+01 8.510e-01
 2.800e+01 1.000e+00]
[3.000e+00 8.800e+01 5.800e+01 1.100e+01 5.400e+01 2.480e+01 2.670e-01
 2.200e+01 0.000e+00]
[6.000e+00 9.200e+01 9.200e+01 0.000e+00 0.000e+00 1.990e+01 1.880e-01
 2.800e+01 0.000e+00]
[1.000e+01 1.220e+02 7.800e+01 3.100e+01 0.000e+00 2.760e+01 5.120e-01
 4.500e+01 0.000e+00]
[4.000e+00 1.030e+02 6.000e+01 3.300e+01 1.920e+02 2.400e+01 9.660e-01
 3.300e+01 0.000e+00]
[1.100e+01 1.380e+02 7.600e+01 0.000e+00 0.000e+00 3.320e+01 4.200e-01
 3.500e+01 0.000e+00]
[9.000e+00 1.020e+02 7.600e+01 3.700e+01 0.000e+00 3.290e+01 6.650e-01
 4.600e+01 1.000e+00]
[2.000e+00 9.000e+01 6.800e+01 4.200e+01 0.000e+00 3.820e+01 5.030e-01
 2.700e+01 1.000e+00]
[4.000e+00 1.110e+02 7.200e+01 4.700e+01 2.070e+02 3.710e+01 1.390e+00
 5.600e+01 1.000e+00]
[3.000e+00 1.800e+02 6.400e+01 2.500e+01 7.000e+01 3.400e+01 2.710e-01
 2.600e+01 0.000e+00]
[7.000e+00 1.330e+02 8.400e+01 0.000e+00 0.000e+00 4.020e+01 6.960e-01
 3.700e+01 0.000e+00]
[7.000e+00 1.060e+02 9.200e+01 1.800e+01 0.000e+00 2.270e+01 2.350e-01
 4.800e+01 0.000e+00]
[9.000e+00 1.710e+02 1.100e+02 2.400e+01 2.400e+02 4.540e+01 7.210e-01
 5.400e+01 1.000e+00]
[7.000e+00 1.590e+02 6.400e+01 0.000e+00 0.000e+00 2.740e+01 2.940e-01
 4.000e+01 0.000e+00]
[0.000e+00 1.800e+02 6.600e+01 3.900e+01 0.000e+00 4.200e+01 1.893e+00
 2.500e+01 1.000e+00]
[1.000e+00 1.460e+02 5.600e+01 0.000e+00 0.000e+00 2.970e+01 5.640e-01
 2.900e+01 0.000e+00]
[2.000e+00 7.100e+01 7.000e+01 2.700e+01 0.000e+00 2.800e+01 5.860e-01
 2.200e+01 0.000e+00]
[7.000e+00 1.030e+02 6.600e+01 3.200e+01 0.000e+00 3.910e+01 3.440e-01
 3.100e+01 1.000e+00]
[7.000e+00 1.050e+02 0.000e+00 0.000e+00 0.000e+00 0.000e+00 3.050e-01
 2.400e+01 0.000e+00]
[1.000e+00 1.030e+02 8.000e+01 1.100e+01 8.200e+01 1.940e+01 4.910e-01
 2.200e+01 0.000e+00]
[1.000e+00 1.010e+02 5.000e+01 1.500e+01 3.600e+01 2.420e+01 5.260e-01
 2.600e+01 0.000e+00]
[5.000e+00 8.800e+01 6.600e+01 2.100e+01 2.300e+01 2.440e+01 3.420e-01
 3.000e+01 0.000e+00]
[8.000e+00 1.760e+02 9.000e+01 3.400e+01 3.000e+02 3.370e+01 4.670e-01
 5.800e+01 1.000e+00]
[7.000e+00 1.500e+02 6.600e+01 4.200e+01 3.420e+02 3.470e+01 7.180e-01
 4.200e+01 0.000e+00]
[1.000e+00 7.300e+01 5.000e+01 1.000e+01 0.000e+00 2.300e+01 2.480e-01
 2.100e+01 0.000e+00]
[7.000e+00 1.870e+02 6.800e+01 3.900e+01 3.040e+02 3.770e+01 2.540e-01
 4.100e+01 1.000e+00]
[0.000e+00 1.000e+02 8.800e+01 6.000e+01 1.100e+02 4.680e+01 9.620e-01
 3.100e+01 0.000e+00]
[0.000e+00 1.460e+02 8.200e+01 0.000e+00 0.000e+00 4.050e+01 1.781e+00
 4.400e+01 0.000e+00]
```

```
 [0.000e+00 1.050e+02 6.400e+01 4.100e+01 1.420e+02 4.150e+01 1.730e-01
  2.200e+01 0.000e+00]
 [2.000e+00 8.400e+01 0.000e+00 0.000e+00 0.000e+00 0.000e+00 3.040e-01
  2.100e+01 0.000e+00]
 [8.000e+00 1.330e+02 7.200e+01 0.000e+00 0.000e+00 3.290e+01 2.700e-01
  3.900e+01 1.000e+00]
 [5.000e+00 4.400e+01 6.200e+01 0.000e+00 0.000e+00 2.500e+01 5.870e-01
  3.600e+01 0.000e+00]
 [2.000e+00 1.410e+02 5.800e+01 3.400e+01 1.280e+02 2.540e+01 6.990e-01
  2.400e+01 0.000e+00]
 [7.000e+00 1.140e+02 6.600e+01 0.000e+00 0.000e+00 3.280e+01 2.580e-01
  4.200e+01 1.000e+00]
 [5.000e+00 9.900e+01 7.400e+01 2.700e+01 0.000e+00 2.900e+01 2.030e-01
  3.200e+01 0.000e+00]
 [0.000e+00 1.090e+02 8.800e+01 3.000e+01 0.000e+00 3.250e+01 8.550e-01
  3.800e+01 1.000e+00]
 [2.000e+00 1.090e+02 9.200e+01 0.000e+00 0.000e+00 4.270e+01 8.450e-01
  5.400e+01 0.000e+00]
 [1.000e+00 9.500e+01 6.600e+01 1.300e+01 3.800e+01 1.960e+01 3.340e-01
  2.500e+01 0.000e+00]
 [4.000e+00 1.460e+02 8.500e+01 2.700e+01 1.000e+02 2.890e+01 1.890e-01
  2.700e+01 0.000e+00]
 [2.000e+00 1.000e+02 6.600e+01 2.000e+01 9.000e+01 3.290e+01 8.670e-01
  2.800e+01 1.000e+00]
 [5.000e+00 1.390e+02 6.400e+01 3.500e+01 1.400e+02 2.860e+01 4.110e-01
  2.600e+01 0.000e+00]
 [1.300e+01 1.260e+02 9.000e+01 0.000e+00 0.000e+00 4.340e+01 5.830e-01
  4.200e+01 1.000e+00]
 [4.000e+00 1.290e+02 8.600e+01 2.000e+01 2.700e+02 3.510e+01 2.310e-01
  2.300e+01 0.000e+00]
 [1.000e+00 7.900e+01 7.500e+01 3.000e+01 0.000e+00 3.200e+01 3.960e-01
  2.200e+01 0.000e+00]
 [1.000e+00 0.000e+00 4.800e+01 2.000e+01 0.000e+00 2.470e+01 1.400e-01
  2.200e+01 0.000e+00]
 [7.000e+00 6.200e+01 7.800e+01 0.000e+00 0.000e+00 3.260e+01 3.910e-01
  4.100e+01 0.000e+00]
 [5.000e+00 9.500e+01 7.200e+01 3.300e+01 0.000e+00 3.770e+01 3.700e-01
  2.700e+01 0.000e+00]
 [0.000e+00 1.310e+02 0.000e+00 0.000e+00 0.000e+00 4.320e+01 2.700e-01
  2.600e+01 1.000e+00]
 [2.000e+00 1.120e+02 6.600e+01 2.200e+01 0.000e+00 2.500e+01 3.070e-01
  2.400e+01 0.000e+00]]
```

In [613…

```
print(training)
```

```
[[3.00e+00 1.13e+02 4.40e+01 ... 1.40e-01 2.20e+01 0.00e+00]
 [2.00e+00 7.40e+01 0.00e+00 ... 1.02e-01 2.20e+01 0.00e+00]
 [7.00e+00 8.30e+01 7.80e+01 ... 7.67e-01 3.60e+01 0.00e+00]
 ...
 [5.00e+00 1.21e+02 7.20e+01 ... 2.45e-01 3.00e+01 0.00e+00]
 [1.00e+00 1.26e+02 6.00e+01 ... 3.49e-01 4.70e+01 1.00e+00]
 [1.00e+00 9.30e+01 7.00e+01 ... 3.15e-01 2.30e+01 0.00e+00]]
```

d.Use inbuilt function to do splitting and interpret the results

In [614...

```
from sklearn.model_selection import train_test_split
arr=data.values
X=arr[:,0:8]
Y=arr[:,8]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20)
print( X_test)
```

```
[[  1.     95.     82.    ...  35.      0.233  43.   ]
 [  0.    128.     68.    ...  30.5     1.391  25.   ]
 [  0.     74.     52.    ...  27.8     0.269  22.   ]
 ...
 [  8.    126.     88.    ...  38.5     0.349  49.   ]
 [  2.    112.     78.    ...  39.4     0.175  24.   ]
 [  1.     97.     66.    ...  23.2     0.487  22.   ]]
```

e.Do normalization of training as well as testing dataset using StandardScaler

In [ ]:

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler().fit(X_test)
scaler=StandardScaler().fit(X_train)
rescaledX=scaler.transform(X)
np.set_printoptions(precision=3)
print(rescaledX[0:2,:])
print(X[0:2,:])

import matplotlib.pyplot as plt
import pandas
mydataframe = pandas.DataFrame(arr)
print(mydataframe)
mydataframe.plot(kind='bar')
plt.show()

from sklearn import preprocessing
fl_x=mydataframe.values.astype(float)
min_max_scaler=preprocessing.MinMaxScaler()
X_scaled=min_max_scaler.fit_transform(fl_x)
df_normalized=pandas.DataFrame(X_scaled)
print(df_normalized)
df_normalized.plot(kind='bar')
plt.show()
```

```
[[ 0.627  0.839  0.165  0.925 -0.698  0.187  0.455  1.451]
 [-0.847 -1.13  -0.142  0.537 -0.698 -0.713 -0.359 -0.187]]
[[  6.    148.     72.     35.      0.     33.6     0.627  50.    ]
 [  1.     85.     66.     29.      0.     26.6     0.351  31.    ]]
        0      1     2     3      4     5      6     7    8
0     6.0  148.0  72.0  35.0    0.0  33.6  0.627  50.0  1.0
1     1.0   85.0  66.0  29.0    0.0  26.6  0.351  31.0  0.0
2     8.0  183.0  64.0   0.0    0.0  23.3  0.672  32.0  1.0
3     1.0   89.0  66.0  23.0   94.0  28.1  0.167  21.0  0.0
4     0.0  137.0  40.0  35.0  168.0  43.1  2.288  33.0  1.0
..    ...    ...   ...   ...    ...   ...    ...   ...  ...
763  10.0  101.0  76.0  48.0  180.0  32.9  0.171  63.0  0.0
764   2.0  122.0  70.0  27.0    0.0  36.8  0.340  27.0  0.0
765   5.0  121.0  72.0  23.0  112.0  26.2  0.245  30.0  0.0
766   1.0  126.0  60.0   0.0    0.0  30.1  0.349  47.0  1.0
767   1.0   93.0  70.0  31.0    0.0  30.4  0.315  23.0  0.0

[768 rows x 9 columns]
```

Is it required to execute the following code for X_test too?

Ans: Yes you need to apply normalisation to test data, if your algorithm works with or needs normalised training data.

That is because your model works on the representation given by its input vectors. The scale of those numbers is part of the representation. This is a bit like converting between feet and metres . . . a model or formula would work with just one type of unit normally.

f.Invoke inbuilt KNN function

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(y_pred)
```

g.Evaluate KNN function

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
matrix =confusion_matrix(y_test, y_pred)
print(confusion_matrix(y_test, y_pred))
accuracy = accuracy_score (y_test,y_pred)
print(classification_report(y_test, y_pred))
```

Explain the output obtained

h.Find the total no of correct predictions

In [ ]:

```
total_test = len(X_test)
total_correct = matrix[0,0]
print(f"Number correct outcome from the total test size of {total_test} is {total_c
```

i.Repeat f,g,h for different values of k in KNN and plot the graph

test 1: value higher than before

In [ ]:
```
error_rate = []
for i in range(1,40):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='black', linestyle='dashed', marker='+',
markerfacecolor='yellow', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```