Name : Anuvind MP

Roll no : AM.EN.U4AIE22010

1. Create two vectors using NumPy and check how many values are equal in the two vectors.

   Example

   V1 = [1 6 7 9]

   V2 = [1 0 6 9]

   Here, the output should be 2.

   (Use np. sum(V1==V2)

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

```
V1 = np.array([1, 6, 7, 9])
V2 = np.array([1, 0, 6, 9])

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

```
2
```

2. Matrix creation using NumPy

   a. Create a matrix M with 10 rows and 3 columns and populate with random values.

   Example:

   [[60 97 34]
   [66 37
   65]
   …..
   [64 64
   44]]

   b. Print size of M. (M.shape)
   c. Print only the number of rows of M(M.shape[0])
   d. Print only the number of columns of M
   e. Write a simple loop to modify the third column as follows: If the sum of the first two columns is divisible by 4, Y should be 1 else, 0.

```
#a
M = np.random.randint(100, size = (10,3))
print(M)
```

```
[[18 62  4]
 [31 55 95]
 [ 7 85 13]
 [40 78 73]
 [92 25 92]
 [82 22 89]
 [32 72 23]
 [ 7 27 69]
 [29 99 57]
 [51 26 56]]
```

```
#b
print("Size of the matrix :", M.shape)
#c
print("Number of rows :", M.shape[0])
#d
print("Number of columns :", M.shape[1])
```

```
Size of the matrix : (10, 3)
Number of rows : 10
Number of columns : 3
```

```
#e
for i in range(M.shape[0]):
  if (M[i][0] + M[i][1]) % 4 == 0:
    M[i][2] = 1
  else:
    M[i][2] = 0

print(M)
```

```
[[18 62  1]
 [31 55  0]
 [ 7 85  1]
 [40 78  0]
 [92 25  0]
 [82 22  1]
 [32 72  1]
 [ 7 27  0]
 [29 99  1]
 [51 26  0]]
```

3. Create pandas dataframe ‘df’ from the created matrix M and name the columns as X1, X2, and Y. (Refer Lab1)

```
df = pd.DataFrame(data=M, columns=["X1", "X2", "Y"])
df
```

	X1	X2	Y
0	18	62	1
1	31	55	0
2	7	85	1
3	40	78	0
4	92	25	0
5	82	22	1
6	32	72	1
7	7	27	0
8	29	99	1
9	51	26	0

Next steps: Generate code with df View recommended plots

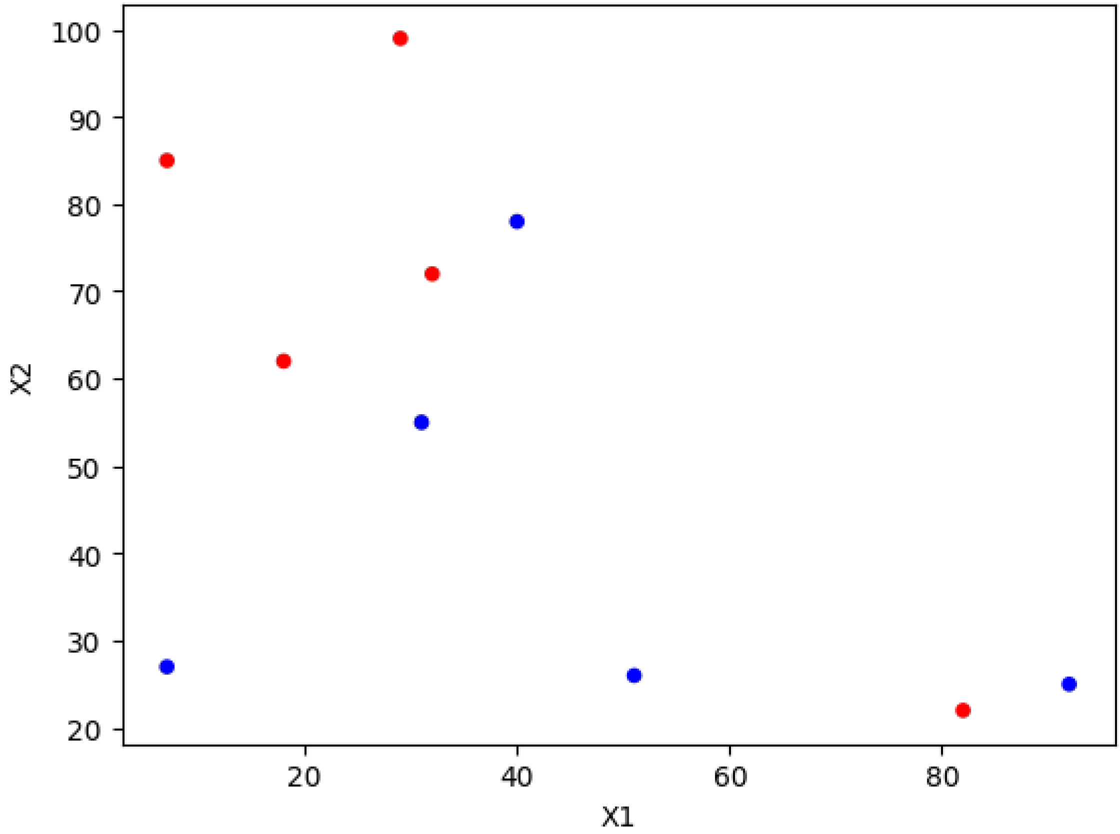
4. Plot X1 and X2 using scatter plot. Color (X1, X2) red if the corresponding Y is 1 else, blue.

```
col = df.Y.map({0:'b', 1:'r'})          #df is the dataframe you created for Q.3
df.plot.scatter(x='X1', y='X2', c=col)
plt.show()
```

4. Plot X1 and X2 using scatter plot. Color (X1, X2) red if the corresponding Y is 1 else, blue.

```
color = df.Y.map({0:'b', 1:'r'})
df.plot.scatter('X1', 'X2', c=color)
```

```
<Axes: xlabel='X1', ylabel='X2'>
```



5.

a. For two columns X1, X2, find squared error: (x1 – x2)^2
(Use np.square)

Example: Matrix M will have
[1369 841 ….. 0]

b. Find the sum of the squared error.
(Use
np.sum)

```
np.square(df['X1'] - df['X2'])
```

```
0    1936
1     576
2    6084
3    1444
4    4489
5    3600
6    1600
7     400
8    4900
9     625
dtype: int64
```

```
np.sum(np.square(df['X1'] - df['X2']))
```

```
25654
```

6. Find Euclidean distance between the first two rows of matrix M.
Compare the result with the inbuilt function numpy.linalg.norm(a-b), where a is the first row and b is the second row.

```
row1 = M[0]
row2 = M[1]
row1, row2
```

```
(array([18, 62,  1]), array([31, 55,  0]))
```

```
def EuclideanDist(a, b):
  return np.sqrt(np.sum(np.square(b-a)))
EuclideanDist(row1, row2)
```

```
14.798648586948742
```

```
np.linalg.norm(row1-row2)
```

```
14.798648586948742
```

7. Create a vector V with three random values. Find the Euclidean distance between each row of M with V. Store the distance in a vector and print.

```
V = np.random.randint(0, 100, 3)
disp(V)
```

$$\begin{bmatrix}79 & 94 & 15\end{bmatrix}$$

```
dist = np.zeros(10, dtype=int)
```

```
for ind, i in enumerate(M):
  dist[ind] = EuclideanDist(i, V)
disp(dist)
```

$$\begin{bmatrix}70 & 63 & 73 & 44 & 71 & 73 & 53 & 99 & 52 & 75\end{bmatrix}$$

8. Create a matrix A with 10 rows and 2 columns. Add a new column to a matrix. (Use np.column_stack). Add a new row to a matrix(Use np.vstack)

```
A=np.array([[1,2,3],[2,3,4]])  print(A)
C=np.array([6,7])
A=np.column_stack((A,C))
print(A)


R=np.array([1,1,1,1])
A=np.vstack((A,R))
print(A)
```

```
A = np.random.randint(0, 100, (10, 2))
disp(A)
```

$$\begin{bmatrix} 93 & 20 \\ 76 & 26 \\ 79 & 37 \\ 63 & 36 \\ 41 & 53 \\ 74 & 56 \\ 85 & 21 \\ 74 & 90 \\ 13 & 46 \\ 32 & 93 \end{bmatrix}$$

```
A = np.column_stack((A, np.random.randint(0, 100, (10, 1))))
disp(A)
```

$$\begin{bmatrix} 93 & 20 & 31 \\ 76 & 26 & 11 \\ 79 & 37 & 57 \\ 63 & 36 & 0 \\ 41 & 53 & 55 \\ 74 & 56 & 9 \\ 85 & 21 & 36 \\ 74 & 90 & 83 \\ 13 & 46 & 24 \\ 32 & 93 & 65 \end{bmatrix}$$

```
A = np.vstack((A, np.random.randint(0, 100, 3)))
disp(A)
```

$$\begin{bmatrix} 93 & 20 & 31 \\ 76 & 26 & 11 \\ 79 & 37 & 57 \\ 63 & 36 & 0 \\ 41 & 53 & 55 \\ 74 & 56 & 9 \\ 85 & 21 & 36 \\ 74 & 90 & 83 \\ 13 & 46 & 24 \\ 32 & 93 & 65 \\ 14 & 52 & 51 \end{bmatrix}$$

9. Create a matrix M1 with two columns X1' and X2' and populate with random values. Find the Euclidean distance between each row of M1 with each row of M. Store the distance in a matrix Dist with 3 columns. The first column is the row id of M, the second column is the row id of M1, and the third column is the distance value. Compare the result with the following code

```
M1 = np.random.randint(0, 100, (10, 2))
disp(M1)
```

$$\begin{bmatrix} 7 & 25 \\ 79 & 95 \\ 60 & 21 \\ 55 & 7 \\ 27 & 91 \\ 45 & 47 \\ 80 & 72 \\ 52 & 0 \\ 62 & 0 \\ 74 & 46 \end{bmatrix}$$

```
Dist = []
for i in range(10):
  for j in range(10):
    Dist.append([i, j, EuclideanDist(M[:, 0:2][i], M1[j])])
Dist = np.array(Dist)
disp(Dist)
```

0.0	0.0	38.6005181312376
0.0	1.0	69.3541635375988
0.0	2.0	58.6941223633168
0.0	3.0	66.2872536767062
0.0	4.0	30.364452901378
0.0	5.0	30.886890422961
0.0	6.0	62.8012738724303
0.0	7.0	70.7106781186548
0.0	8.0	76.0263112349928
0.0	9.0	58.2408791142441
1.0	0.0	38.4187454245971
1.0	1.0	62.4819974072532
1.0	2.0	44.6878059430087
1.0	3.0	53.665631459995
1.0	4.0	36.2215405525497
1.0	5.0	16.1245154965971
1.0	6.0	51.8652099195598
1.0	7.0	58.8727441181401
1.0	8.0	63.1347764706584
1.0	9.0	43.9317652729776
2.0	0.0	60.0
2.0	1.0	72.691127381545
2.0	2.0	83.0963296421689
2.0	3.0	91.5860251348425
2.0	4.0	20.8806130178211
2.0	5.0	53.7401153701776
2.0	6.0	74.1484996476665
2.0	7.0	96.1769203083567
2.0	8.0	101.242283656583
2.0	9.0	77.5241897732572
3.0	0.0	62.4339651151519
3.0	1.0	42.5440947723653
3.0	2.0	60.4069532421558
3.0	3.0	72.5672102261069
3.0	4.0	18.3847763108502
3.0	5.0	31.4006369362152
3.0	6.0	40.4474968323134
3.0	7.0	78.9176786277954
3.0	8.0	81.0431983574192
3.0	9.0	46.690470119715
4.0	0.0	85.0
4.0	1.0	71.1969100453102
4.0	2.0	32.2490309931942
4.0	3.0	41.146081222882
4.0	4.0	92.633687176966
4.0	5.0	51.8941229813165
4.0	6.0	48.5077313425396
4.0	7.0	47.169905660283
4.0	8.0	39.0512483795333
4.0	9.0	27.6586333718787
5.0	0.0	75.0599760191808

5.0	1.0	73.0616178304313
5.0	2.0	22.0227155455452
5.0	3.0	30.886890422961
5.0	4.0	88.2383136738231
5.0	5.0	44.6542271235322
5.0	6.0	50.0399840127872
5.0	7.0	37.2021504754766
5.0	8.0	29.732137494637
5.0	9.0	25.298221281347
6.0	0.0	53.2353266168247
6.0	1.0	52.3259018078045
6.0	2.0	58.1807528311554
6.0	3.0	68.9492567037528
6.0	4.0	19.6468827043885
6.0	5.0	28.1780056072107
6.0	6.0	48.0
6.0	7.0	74.7261667690776
6.0	8.0	78.0
6.0	9.0	49.3963561409139
7.0	0.0	2.0
7.0	1.0	99.0353472251196
7.0	2.0	53.3385414123783
7.0	3.0	52.0
7.0	4.0	67.0522184569608
7.0	5.0	42.9418211071678
7.0	6.0	85.7554662980734
7.0	7.0	52.4785670536077
7.0	8.0	61.2698947281616
7.0	9.0	69.6419413859206
8.0	0.0	77.2010362624751
8.0	1.0	50.1597448159378
8.0	2.0	83.9344982709732
8.0	3.0	95.603347221737
8.0	4.0	8.24621125123532
8.0	5.0	54.4058820349418
8.0	6.0	57.706152185014
8.0	7.0	101.636607578175
8.0	8.0	104.355162785557
8.0	9.0	69.5269731830748
9.0	0.0	44.0113621693308
9.0	1.0	74.4647567645259
9.0	2.0	10.295630140987
9.0	3.0	19.4164878389476
9.0	4.0	69.28924880528
9.0	5.0	21.8403296678416
9.0	6.0	54.3783044972901
9.0	7.0	26.0192236625154
9.0	8.0	28.2311884269862
9.0	9.0	30.4795013082563

```
from sklearn.metrics.pairwise import euclidean_distances
dist10x10 = euclidean_distances(M[:, 0:2], M1)
disp(dist10x10)
```

$$\begin{bmatrix} 38.6005181312376 & 69.3541635375988 & 58.6941223633168 & 66.2872536767062 & 30.364 \\ 38.4187454245971 & 62.4819974072532 & 44.6878059430087 & 53.665631459995 & 36.221 \\ 60.0 & 72.691127381545 & 83.0963296421689 & 91.5860251348425 & 20.880 \\ 62.4339651151519 & 42.5440947723653 & 60.4069532421558 & 72.5672102261069 & 18.384 \\ 85.0 & 71.1969100453102 & 32.2490309931942 & 41.146081222882 & 92.633 \\ 75.0599760191808 & 73.0616178304313 & 22.0227155455452 & 30.886890422961 & 88.238 \\ 53.2353266168247 & 52.3259018078045 & 58.1807528311554 & 68.9492567037528 & 19.646 \\ 2.0 & 99.0353472251196 & 53.3385414123783 & 52.0 & 67.052 \\ 77.2010362624751 & 50.1597448159378 & 83.9344982709732 & 95.603347221737 & 8.2462 \\ 44.0113621693308 & 74.4647567645259 & 10.295630140987 & 19.4164878389476 & 69.28 \end{bmatrix}$$

```
print(f"{np.sum(dist10x10.reshape((100, 1)) == Dist[:, 2])}% of the Distances are the sam
```

```
102% of the Distances are the same
```

10. Sort the Dist matrix based on the 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.

```
disp(Dist[Dist[:,2].argsort()])
```

$$\begin{bmatrix}
7.0 & 0.0 & 2.0 \\
8.0 & 4.0 & 8.24621125123532 \\
9.0 & 2.0 & 10.295630140987 \\
1.0 & 5.0 & 16.1245154965971 \\
3.0 & 4.0 & 18.3847763108502 \\
9.0 & 3.0 & 19.4164878389476 \\
6.0 & 4.0 & 19.6468827043885 \\
2.0 & 4.0 & 20.8806130178211 \\
9.0 & 5.0 & 21.8403296678416 \\
5.0 & 2.0 & 22.0227155455452 \\
5.0 & 9.0 & 25.298221281347 \\
9.0 & 7.0 & 26.0192236625154 \\
4.0 & 9.0 & 27.6586333718787 \\
6.0 & 5.0 & 28.1780056072107 \\
9.0 & 8.0 & 28.2311884269862 \\
5.0 & 8.0 & 29.732137494637 \\
0.0 & 4.0 & 30.364452901378 \\
9.0 & 9.0 & 30.4795013082563 \\
0.0 & 5.0 & 30.886890422961 \\
5.0 & 3.0 & 30.886890422961 \\
3.0 & 5.0 & 31.4006369362152 \\
4.0 & 2.0 & 32.2490309931942 \\
1.0 & 4.0 & 36.2215405525497 \\
5.0 & 7.0 & 37.2021504754766 \\
1.0 & 0.0 & 38.4187454245971 \\
0.0 & 0.0 & 38.6005181312376 \\
4.0 & 8.0 & 39.0512483795333 \\
3.0 & 6.0 & 40.4474968323134 \\
4.0 & 3.0 & 41.146081222882 \\
3.0 & 1.0 & 42.5440947723653 \\
7.0 & 5.0 & 42.9418211071678 \\
1.0 & 9.0 & 43.9317652729776 \\
9.0 & 0.0 & 44.0113621693308 \\
5.0 & 5.0 & 44.6542271235322 \\
1.0 & 2.0 & 44.6878059430087 \\
3.0 & 9.0 & 46.690470119715 \\
4.0 & 7.0 & 47.169905660283 \\
6.0 & 6.0 & 48.0 \\
4.0 & 6.0 & 48.5077313425396 \\
6.0 & 9.0 & 49.3963561409139 \\
5.0 & 6.0 & 50.0399840127872 \\
8.0 & 1.0 & 50.1597448159378 \\
1.0 & 6.0 & 51.8652099195598 \\
4.0 & 5.0 & 51.8941229813165 \\
7.0 & 3.0 & 52.0 \\
6.0 & 1.0 & 52.3259018078045 \\
7.0 & 7.0 & 52.4785670536077 \\
6.0 & 0.0 & 53.2353266168247 \\
7.0 & 2.0 & 53.3385414123783 \\
1.0 & 3.0 & 53.665631459995 \\
2.0 & 5.0 & 53.7401153701776
\end{bmatrix}$$

9.0	6.0	54.3783044972901
8.0	5.0	54.4058820349418
8.0	6.0	57.706152185014
6.0	2.0	58.1807528311554
0.0	9.0	58.2408791142441
0.0	2.0	58.6941223633168
1.0	7.0	58.8727441181401
2.0	0.0	60.0
3.0	2.0	60.4069532421558
7.0	8.0	61.2698947281616
3.0	0.0	62.4339651151519
1.0	1.0	62.4819974072532
0.0	6.0	62.8012738724303
1.0	8.0	63.1347764706584
0.0	3.0	66.2872536767062
7.0	4.0	67.0522184569608
6.0	3.0	68.9492567037528
9.0	4.0	69.28924880528
0.0	1.0	69.3541635375988
8.0	9.0	69.5269731830748
7.0	9.0	69.6419413859206
0.0	7.0	70.7106781186548
4.0	1.0	71.1969100453102
3.0	3.0	72.5672102261069
2.0	1.0	72.691127381545
5.0	1.0	73.0616178304313
2.0	6.0	74.1484996476665
9.0	1.0	74.4647567645259
6.0	7.0	74.7261667690776
5.0	0.0	75.0599760191808
0.0	8.0	76.0263112349928
8.0	0.0	77.2010362624751
2.0	9.0	77.5241897732572
6.0	8.0	78.0
3.0	7.0	78.9176786277954
3.0	8.0	81.0431983574192
2.0	2.0	83.0963296421689
8.0	2.0	83.9344982709732
4.0	0.0	85.0
7.0	6.0	85.7554662980734
5.0	4.0	88.2383136738231
2.0	3.0	91.5860251348425
4.0	4.0	92.633687176966
8.0	3.0	95.603347221737
2.0	7.0	96.1769203083567
7.0	1.0	99.0353472251196
2.0	8.0	101.242283656583
8.0	7.0	101.636607578175
8.0	8.0	104.355162785557

11. Get the initial k rows from the sorted matrix

```
k = int(input("Enter the k rows to be Extracted: "))
disp(Dist[Dist[:,2].argsort()][0:k, :])
```

```
Enter the k rows to be Extracted: 6
```

$$\begin{bmatrix} 7.0 & 0.0 & 2.0 \\ 8.0 & 4.0 & 8.24621125123532 \\ 9.0 & 2.0 & 10.295630140987 \\ 1.0 & 5.0 & 16.1245154965971 \\ 3.0 & 4.0 & 18.3847763108502 \\ 9.0 & 3.0 & 19.4164878389476 \end{bmatrix}$$

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

```
sortedDistk = Dist[Dist[:,2].argsort()][0:k, :]
one = np.sum(sortedDistk[:, 0] == 1)
zero = np.sum(sortedDistk[:, 0] == 0)
result = 1 if one > zero else 0
print(result)
```

```
1
```

PART B

PART B : KNN implementation

a. Load diabetes dataset as done in Lab 1.
b. Peek at a few rows as done in Lab 1
c. Split the dataset into 80% training and 20% testing using numpy slicing.
d. Use the inbuilt function to do splitting and interpret results

```
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)
```

e. Do normalisation of training as well as testing dataset using StandardScaler as done in Lab 1. Is it required to execute the following code for X_test, too?

```
scaler=StandardScaler().fit(X_train)
```

f. Invoke inbuilt kNN function.

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

Interpret the output obtained.

g. Evaluate kNN

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Explain the output obtained.
h. Find the total number of correct predictions.
i. Repeat f, g, h for different values of k in kNN. And plot the graph.

```
df = pd.read_csv("/content/diabetes_dataset.csv")
```

b

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness
0	6	148	72	35
1	1	85	66	29
2	8	183	64	0
3	1	89	66	23
4	0	137	40	35

Next steps: Generate code with df View recommended plots

c

```
X = df.drop("Outcome", axis=1)
X
```

	Pregnancies	Glucose	BloodPressure	SkinThickne
0	6	148	72	
1	1	85	66	
2	8	183	64	
3	1	89	66	
4	0	137	40	
...	...	...	...	
763	10	101	76	
764	2	122	70	
765	5	121	72	
766	1	126	60	
767	1	93	70	

768 rows × 8 columns

Next steps: Generate code with X View recommended plots

```
y = df["Outcome"]
y
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

d

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
print(X_test)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
558           11      103             68             40        0  46.2
650            1       91             54             25      100  25.2
765            5      121             72             23      112  26.2
675            6      195             70              0        0  30.9
60             2       84              0              0        0   0.0
..           ...      ...            ...            ...      ...   ...
376            0       98             82             15       84  25.2
317            3      182             74              0        0  30.5
62             5       44             62              0        0  25.0
660           10      162             84              0        0  27.7
15             7      100              0              0        0  30.0

     DiabetesPedigreeFunction  Age
558                     0.126   42
650                     0.234   23
765                     0.245   30
675                     0.328   31
60                      0.304   21
..                        ...  ...
376                     0.299   22
317                     0.345   29
62                      0.587   36
660                     0.182   54
15                      0.484   32

[154 rows x 8 columns]
```

e

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler().fit(X_train)
```

```
X_train =scaler.transform(X_train)
X_train
```

```
array([[-0.51311531,  0.42552711,  0.07360599, ..., -0.38696797,
         0.19651613, -0.84904579],
       [-0.51311531, -0.62389236, -0.56744474, ..., -0.98020648,
         0.40967178, -0.84904579],
       [-1.10699429, -0.94189826, -0.0332358 , ...,  1.00146258,
        -0.2801236 , -0.678347  ],
       ...,
       [ 1.26852163,  0.10752121,  0.39413135, ..., -0.41221216,
         0.62578793,  1.62608671],
       [-1.10699429, -0.87829708, -0.46060295, ...,  0.42084617,
        -0.62946203, -0.678347  ],
       [-0.8100548 ,  0.83893478,  1.35570744, ...,  2.1879396 ,
        -0.34821499, -0.5076482 ]])
```

```
X_test = scaler.transform(X_test)
X_test
```

```
array([[ 2.1593401 , -0.56029118, -0.0332358 , ...,  1.79665462,
        -1.03504987,  0.77259275],
       [-0.8100548 , -0.94189826, -0.78112831, ..., -0.85398552,
        -0.71531639, -0.84904579],
       [ 0.37770316,  0.01211944,  0.18044777, ..., -0.72776456,
        -0.68275095, -0.25160001],
       ...,
       [ 0.37770316, -2.43652598, -0.35376116, ..., -0.87922971,
         0.32973841,  0.26049637],
       [ 1.86240061,  1.31594363,  0.8214985 , ..., -0.53843312,
        -0.86926214,  1.79678551],
       [ 0.97158214, -0.65569295, -3.66585658, ..., -0.24812491,
         0.02480741, -0.08090122]])
```

f

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

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

g

```
from sklearn.metrics import classification_report, confusion_matrix
confus = confusion_matrix(y_test, y_pred)
print(confus)
print(classification_report(y_test, y_pred))
```

```
[[84 18]
 [21 31]]
              precision    recall  f1-score   support

           0       0.80      0.82      0.81       102
           1       0.63      0.60      0.61        52

    accuracy                           0.75       154
   macro avg       0.72      0.71      0.71       154
weighted avg       0.74      0.75      0.74       154
```

h

```
confus[0][0] + confus[1][1]
```

```
115
```

i

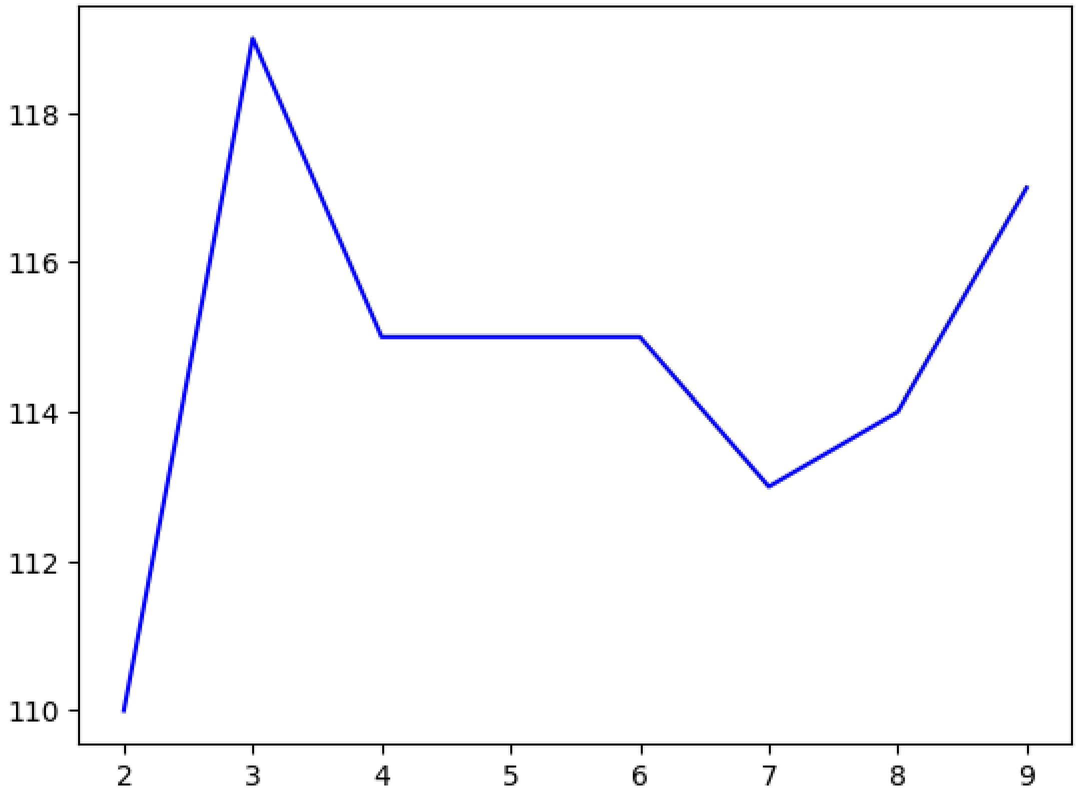
```
points = []
for i in range(2, 10):
  classifier = KNeighborsClassifier(n_neighbors=i)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  confus = confusion_matrix(y_test, y_pred)
  points.append([i, confus[0][0] + confus[1][1]])
disp(np.array(points))
```

$$\begin{bmatrix} 2 & 110 \\ 3 & 119 \\ 4 & 115 \\ 5 & 115 \\ 6 & 115 \\ 7 & 113 \\ 8 & 114 \\ 9 & 117 \end{bmatrix}$$

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
plt.plot(np.array(points)[:, 0], np.array(points)[:, 1], 'b-')
```

```
[<matplotlib.lines.Line2D at 0x7eab96834f70>]
```



Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

```
Start coding or generate with AI.
```