Lab Assignment 2
(Refer to NumPy tutorial)

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]

```
import numpy as np
import pandas as pd
from sympy import Array as disp
%matplotlib inline
```

```
def eqsum(arr1, arr2):
  return np.sum(arr1 == arr2)
```

```
arr1 = np.array([1, 6, 7, 9])
arr2 = np.array([1, 0, 6, 9])
disp(arr2)
```

$$\begin{bmatrix}1 & 0 & 6 & 9\end{bmatrix}$$

```
eqsum(arr1, arr2)
```

```
2
```

2. Matrix creation using NumPy

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

   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.

```
M = np.random.randint(0, 100, (10, 3))
```

```
disp(M)
```

$$\begin{bmatrix}62 & 13 & 56\\35 & 38 & 88\\17 & 14 & 56\\97 & 98 & 15\\37 & 61 & 29\\28 & 47 & 1\\3 & 74 & 77\\33 & 23 & 13\\34 & 33 & 11\\33 & 57 & 95\end{bmatrix}$$

```
M.shape
```

```
(10, 3)
```

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

```
10
```

```
M.shape[1]
```

```
3
```

```
for i in M:
  if i[0] + i[1] % 4 == 0:
    i[2] = 1
  else:
    i[2]=0
disp(M)
```

$$\begin{bmatrix} 62 & 13 & 0 \\ 35 & 38 & 0 \\ 17 & 14 & 0 \\ 97 & 98 & 0 \\ 37 & 61 & 0 \\ 28 & 47 & 0 \\ 3 & 74 & 0 \\ 33 & 23 & 0 \\ 34 & 33 & 0 \\ 33 & 57 & 0 \end{bmatrix}$$

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	62	13	0
1	35	38	0
2	17	14	0
3	97	98	0
4	37	61	0
5	28	47	0
6	3	74	0
7	33	23	0
8	34	33	0
9	33	57	0

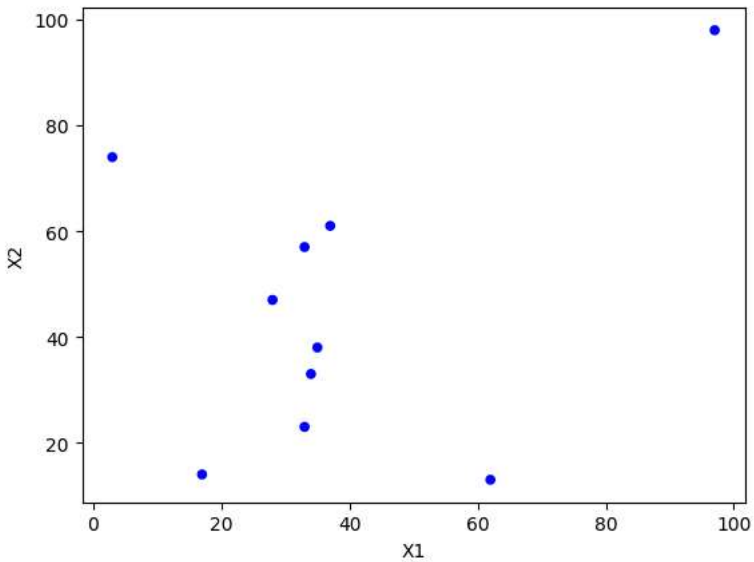
Next steps: Generate code with df View recommended plots

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()
```

```
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
.sum)

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

```
0    2401
1       9
2       9
3       1
4     576
5     361
6    5041
7     100
8       1
9     576
dtype: int64
```

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

```
9075
```

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([62, 13,  0]), array([35, 38,  0]))
```

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

```
36.796738985948195
```

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

```
36.796738985948195
```

first row and b is the second row.

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)
```

$$\begin{bmatrix} 62 & 47 & 34 \end{bmatrix}$$

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

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

$$\begin{bmatrix} 48 & 44 & 65 & 70 & 44 & 48 & 73 & 50 & 46 & 45 \end{bmatrix}$$

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)
```

$$\begin{bmatrix} 28 & 82 \\ 64 & 68 \\ 43 & 94 \\ 53 & 85 \\ 87 & 89 \\ 68 & 67 \\ 92 & 3 \\ 57 & 50 \\ 59 & 0 \\ 65 & 95 \end{bmatrix}$$

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

$$\begin{bmatrix} 28 & 82 & 96 \\ 64 & 68 & 37 \\ 43 & 94 & 17 \\ 53 & 85 & 43 \\ 87 & 89 & 44 \\ 68 & 67 & 80 \\ 92 & 3 & 87 \\ 57 & 50 & 28 \\ 59 & 0 & 50 \\ 65 & 95 & 0 \end{bmatrix}$$

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

$$\begin{bmatrix} 28 & 82 & 96 \\ 64 & 68 & 37 \\ 43 & 94 & 17 \\ 53 & 85 & 43 \\ 87 & 89 & 44 \\ 68 & 67 & 80 \\ 92 & 3 & 87 \\ 57 & 50 & 28 \\ 59 & 0 & 50 \\ 65 & 95 & 0 \\ 25 & 27 & 13 \end{bmatrix}$$

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)
```

$$\begin{bmatrix} 93 & 64 \\ 98 & 27 \\ 55 & 89 \\ 95 & 30 \\ 5 & 84 \\ 76 & 56 \\ 33 & 96 \\ 26 & 60 \\ 15 & 74 \\ 13 & 82 \end{bmatrix}$$

```
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)
```

5.0	1.0	72.8010988928052
5.0	2.0	49.9299509312797
5.0	3.0	69.1230786351418
5.0	4.0	43.5660418215839
5.0	5.0	48.8364617882991
5.0	6.0	49.254441424099
5.0	7.0	13.1529464379659
5.0	8.0	29.9666481275434
5.0	9.0	38.0788655293195
6.0	0.0	90.5538513813742
6.0	1.0	105.990565617889
6.0	2.0	54.1202365109392
6.0	3.0	101.980390271856
6.0	4.0	10.1980390271856
6.0	5.0	75.1864349467376
6.0	6.0	37.2021504754766
6.0	7.0	26.9258240356725
6.0	8.0	12.0
6.0	9.0	12.8062484748657
7.0	0.0	72.6704891960967
7.0	1.0	65.1229606206597
7.0	2.0	69.5701085237043
7.0	3.0	62.393909959226
7.0	4.0	67.1192967781993
7.0	5.0	54.2033209314706
7.0	6.0	73.0
7.0	7.0	37.6563407675255
7.0	8.0	54.0832691319598
7.0	9.0	62.2976725086901
8.0	0.0	66.6483308118065
8.0	1.0	64.2806347199528
8.0	2.0	59.8080262172227
8.0	3.0	61.0737259384099
8.0	4.0	58.6685605754905
8.0	5.0	47.8852795752515
8.0	6.0	63.0079360080935
8.0	7.0	28.1602556806574
8.0	8.0	45.18849411078
8.0	9.0	53.3104117410474
9.0	0.0	60.4069532421558
9.0	1.0	71.5891053163818
9.0	2.0	38.8329756778952
9.0	3.0	67.6239602507869
9.0	4.0	38.8973006775534
9.0	5.0	43.0116263352131
9.0	6.0	39.0
9.0	7.0	7.61577310586391
9.0	8.0	24.7588368062799
9.0	9.0	32.0156211871642

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

59.6824932455071	38.6264158316559	76.3216876123687	37.1214223865412	91.04943
63.5609943282828	63.9531078212779	54.7813836992093	60.5309838016862	54.91812
90.9725233243533	82.0365772079747	84.0773453434396	79.6241169495775	71.02112
34.2344855372474	71.0070419043069	42.9534631898291	68.0294054067798	93.05912
56.0802995712398	69.8355210476732	33.2866339541865	65.7647321898295	39.40812
67.1863081289633	72.8010988928052	49.9299509312797	69.1230786351418	43.56604
90.5538513813742	105.990565617889	54.1202365109392	101.980390271856	10.19803
72.6704891960967	65.1229606206597	69.5701085237043	62.393909959226	67.11929
66.6483308118065	64.2806347199528	59.8080262172227	61.0737259384099	58.66856
60.4069532421558	71.5891053163818	38.8329756778952	67.6239602507869	38.89730

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

```
100% 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()])
```

6.0	2.0	54.1202365109392
7.0	5.0	54.2033209314706
1.0	2.0	54.7813836992093
1.0	4.0	54.9181208709839
4.0	0.0	56.0802995712398
1.0	6.0	58.0344725141876
8.0	4.0	58.6685605754905
0.0	7.0	59.2030404624627
0.0	0.0	59.6824932455071
8.0	2.0	59.8080262172227
2.0	8.0	60.0333240792145
9.0	0.0	60.4069532421558
1.0	3.0	60.5309838016862
8.0	3.0	61.0737259384099
7.0	9.0	62.2976725086901
7.0	3.0	62.393909959226
8.0	6.0	63.0079360080935
1.0	0.0	63.5609943282828
1.0	1.0	63.9531078212779
3.0	6.0	64.0312423743285
8.0	1.0	64.2806347199528
7.0	1.0	65.1229606206597
4.0	3.0	65.7647321898295
8.0	0.0	66.6483308118065
7.0	4.0	67.1192967781993
5.0	0.0	67.1863081289633
9.0	3.0	67.6239602507869
3.0	3.0	68.0294054067798
2.0	9.0	68.1175454637056
5.0	3.0	69.1230786351418
7.0	2.0	69.5701085237043
4.0	1.0	69.8355210476732
3.0	1.0	71.0070419043069
2.0	4.0	71.0211236182588
9.0	1.0	71.5891053163818
2.0	5.0	72.4223722339996
7.0	0.0	72.6704891960967
5.0	1.0	72.8010988928052
7.0	6.0	73.0
6.0	5.0	75.1864349467376
0.0	2.0	76.3216876123687
0.0	8.0	77.0064932327138
2.0	3.0	79.6241169495775
3.0	7.0	80.529497701153
2.0	1.0	82.0365772079747
2.0	6.0	83.5463942968217
2.0	2.0	84.0773453434396
0.0	9.0	84.6286003665428
3.0	8.0	85.4400374531753
3.0	9.0	85.5102333057278

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: 10

$$\begin{bmatrix} 9.0 & 7.0 & 7.61577310586391 \\ 6.0 & 4.0 & 10.1980390271856 \\ 4.0 & 7.0 & 11.0453610171873 \\ 6.0 & 8.0 & 12.0 \\ 6.0 & 9.0 & 12.8062484748657 \\ 5.0 & 7.0 & 13.1529464379659 \\ 1.0 & 7.0 & 23.7697286480094 \\ 9.0 & 8.0 & 24.7588368062799 \\ 4.0 & 8.0 & 25.5538646783613 \\ 6.0 & 7.0 & 26.9258240356725 \end{bmatrix}$$

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 : KNN implementation

a. Load diabetes dataset as done in Lab 1.

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

b. Peek at a few rows as done in Lab 1

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
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	

Next steps: Generate code with df View recommended plots

c. Split the dataset into 80% training and 20% testing using numpy slicing.

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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
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	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

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. 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)
```

```
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  \
229            0      117             80             31       53  45.2
245            9      184             85             15        0  30.0
444            4      117             62             12        0  29.7
84             5      137            108              0        0  48.8
179            5      130             82              0        0  39.1
..           ...      ...            ...            ...      ...   ...
338            9      152             78             34      171  34.2
530            2      122             60             18      106  29.8
577            2      118             80              0        0  42.9
157            1      109             56             21      135  25.2
354            3       90             78              0        0  42.7

     DiabetesPedigreeFunction  Age
229                     0.089   24
245                     1.213   49
444                     0.380   30
84                      0.227   37
179                     0.956   37
..                        ...  ...
338                     0.893   33
530                     0.717   22
577                     0.693   21
157                     0.833   23
354                     0.559   21

[154 rows x 8 columns]
```

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)

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

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

```
array([[-0.26087313,  0.89869308,  0.36731248, ..., -1.37327601,
        -0.79315116,  0.32742436],
       [-1.13456095,  1.8597968 ,  0.98562601, ...,  1.39214003,
        -0.74837359, -0.62250009],
       [-0.84333168, -0.06241064,  0.98562601, ...,  1.64015941,
         0.10240011, -0.62250009],
       ...,
       [ 0.03035615, -0.68247756,  0.16120797, ..., -0.80283145,
        -0.53344129, -0.44978656],
       [-1.13456095, -0.65147422,  0.98562601, ...,  1.82617394,
         1.46065287, -0.19071625],
       [ 0.32158542,  0.46464624,  0.67646925, ..., -3.97747946,
         0.49942784,  3.09084095]])
```

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

```
array([[-1.13456095, -0.12441734,  0.57341699, ...,  1.62775844,
        -1.14540132, -0.79521363],
       [ 1.48650252,  1.95280684,  0.83104763, ..., -0.25718882,
         2.20993077,  1.36370558],
       [ 0.03035615, -0.12441734, -0.35405331, ..., -0.29439173,
        -0.27671659, -0.27707302],
       ...,
       [-0.5521024 , -0.09341399,  0.57341699, ...,  1.34253615,
         0.6576419 , -1.05428393],
       [-0.84333168, -0.3724441 , -0.66321008, ..., -0.85243532,
         1.07556583, -0.8815704 ],
       [-0.26087313, -0.96150768,  0.47036473, ...,  1.31773422,
         0.257629  , -1.05428393]])
```