LAB 2

PART A : Prerequisite for kNN implementation.(Q1. – Q.12 may help you to implement kNN on your own)

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

In [108…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("Solarize_Light2")
np.random.seed(9)
np.set_printoptions(suppress=True)
```

In [109…
```
V1 = np.array([1, 6, 7, 9])
V2 = np.array([1, 0, 6, 9])
```

In [110…
```
V1==V2
```

Out[110…
```
array([ True, False, False,  True])
```

In [111…
```
print("Number of values common in V1 & V2: ", np.sum(V1==V2))
```

```
Number of values common in V1 & V2:  2
```

2. Matrix creation using numpy

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

In [112…
```
rng = np.random.default_rng()
```

In [113…
```
M = rng.integers(1000, size=(10, 3))
M
```

Out[113…
```
array([[304, 391, 437],
       [ 99, 424, 258],
       [196, 278, 981],
       [440, 279, 209],
       [269, 124, 934],
       [ 70,  96, 497],
       [421, 340, 501],
       [681, 869, 795],
       [561, 766,  69],
       [337, 785, 383]], dtype=int64)
```

b. Print size of M.

In [114…
```
print("Size of M: ", M.shape)
```

```
Size of M:  (10, 3)
```

c. Print only number of rows of M

In [115…
```
print("Number of rows of M: ", M.shape[0])
```

```
Number of rows of M:  10
```

d. Print only number of columns of M

In [116…
```
print("Number of columns of M: ", M.shape[1])
```

```
Number of columns of M:  3
```

e. Write a simple loop to modify third column as follows: If sum of first two columns is divisible by 4, Y should be 1 else 0.

In [117…
```
for i in range(len(M)):
    s = M[i][0] + M[i][1]
    if( (s % 4) == 0):
        M[i][2] = 1
    else:
        M[i][2] = 0
```

In [118…
```
M
```

Out[118…
```
array([[304, 391,   0],
       [ 99, 424,   0],
       [196, 278,   0],
       [440, 279,   0],
       [269, 124,   0],
       [ 70,  96,   0],
       [421, 340,   0],
       [681, 869,   0],
       [561, 766,   0],
       [337, 785,   0]], dtype=int64)
```

3. Create a pandas dataframe 'df' from the created matrix M and name the columns as X1,X2, and Y.

In [119…
```
df = pd.DataFrame(M, columns = ['X1', 'X2', 'Y'])
df
```

Out[119…

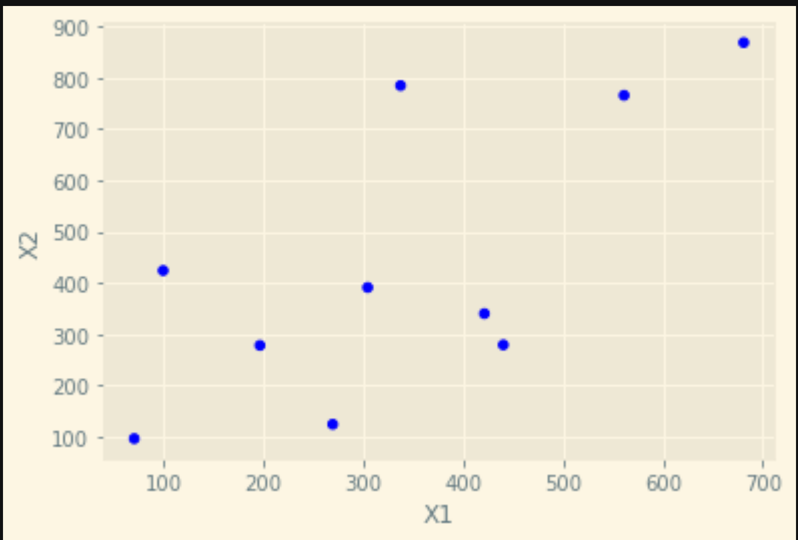
	X1	X2	Y
0	304	391	0
1	99	424	0
2	196	278	0
3	440	279	0
4	269	124	0
5	70	96	0
6	421	340	0
7	681	869	0
8	561	766	0
9	337	785	0

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

1 else blue.

In [120…

```
'''y = M[:,2]
col = np.where(y==1,'r','b')
plt.scatter(M[:,0], M[:,1], c=col)
plt.show()'''
col = df.Y.map({0:'b', 1:'r'})
df.plot.scatter(x='X1', y='X2', c=col)
plt.show()
```



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

In [121…

```
Z = np.square((M[:,0]-M[:,1]))
print("squared error : (x1 - x2)^2 = \n", Z)
```

```
squared error : (x1 - x2)^2 =
 [  7569 105625   6724  25921  21025    676   6561  35344  42025 200704]
```

b. Find sum of the squared error

In [122…

```
Zsum = np.sum(Z)
print("Sum of [squared error : (x1 - x2)^2] = ", Zsum)
```

```
Sum of [squared error : (x1 - x2)^2] =  452174
```

6. Find euclidean distance between first two rows of matrix M.

Compare result with inbuilt function numpy.linalg.norm(a-b) where a is first row and b is second row.

In [123…

```
ed = np.sqrt(np.sum(np.square(M[:,0] - M[:,1])))
print("Euclidean Distance = ", ed)
```

```
Euclidean Distance =  672.4388448030052
```

In [124…

```
print("Euclidean Distance = ", np.linalg.norm(M[:,0]-M[:,1]))
```

```
Euclidean Distance =  672.4388448030052
```

7. Create a vector V with two random values.
Find the Euclidean distance between each row of M with V.

Store the distance in a vector and print

In [125…
```
V = rng.integers(1000, size=(2))
V
```

Out[125…
```
array([464, 555], dtype=int64)
```

In [126…
```
eu_dist = np.linalg.norm(M[:, 0:2] - V, axis=1)
```

In [127…
```
print("Euclidean Distance between single point and matrix M:\n", eu_dist)
```

```
Euclidean Distance between single point and matrix M:
 [229.12005587 387.7963383  385.42573863 277.04151313 473.06024986
 604.91073722 219.25783908 381.68704458 232.22833591 262.73370549]
```

8. Manipulate matrix
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)

In [128…
```
A = rng.integers(1000, size=(10, 2))
print(A)
```

```
[[967 646]
 [992 927]
 [745 548]
 [495 734]
 [446  30]
 [540 239]
 [995   4]
 [399 568]
 [850 994]
 [119 253]]
```

In [129…
```
C = rng.integers(1000, size=10)
print(C)
```

```
[480 339 191 711 507 762 458  55 312 281]
```

In [130…
```
A = np.column_stack((A, C))
print(A)
```

```
[[967 646 480]
 [992 927 339]
 [745 548 191]
 [495 734 711]
 [446  30 507]
 [540 239 762]
 [995   4 458]
 [399 568  55]
 [850 994 312]
 [119 253 281]]
```

In [131…
```
R = rng.integers(1000, size=3)
print(R)
```

```
[552  88 489]
```

In [132…
```
A = np.vstack((A, R))
print(A)
```

```
[[967 646 480]
 [992 927 339]
 [745 548 191]
 [495 734 711]
 [446  30 507]
 [540 239 762]
 [995   4 458]
 [399 568  55]
 [850 994 312]
 [119 253 281]
 [552  88 489]]
```

9. Create a matrix M' with two columns X1', X2' and populate with random values.
Find the Euclidean distance between each row of M' with each row of M(excluding the last column Y) created in Q.2
Store the distance in a matrix Dist with 3 columns. First column is the row id of M, second column is the row id of M', and the third column is distance value

Compare result with following code

In [133…

```
from sklearn.metrics.pairwise import euclidean_distances
```

In [134…

```
M_ = rng.integers(1000, size=(10, 2))
M_
```

Out[134…

```
array([[391, 213],
       [270, 238],
       [ 94, 541],
       [314, 535],
       [297, 103],
       [656, 664],
       [575, 274],
       [933, 624],
       [ 32, 696],
       [211, 254]], dtype=int64)
```

In [135…

```
Dist = np.linalg.norm(M[:,0:2] - M_, axis=1)
Dist
```

Out[135…

```
array([198.12369873, 252.65985039, 282.08686605, 285.32788157,
        35.        , 816.10048401, 167.54700833, 351.4669259 ,
       533.61128174, 545.74444569])
```

In [136…

```
def ed(v1, v2):
    return np.linalg.norm(v1-v2)
```

In [137…

```
M2 = M[:,0:2]
Dist = np.empty((0, 3))
for i in range(len(M_)):
    for j in range(len(M2)):
        d = ed(M_[i], M2[j])
        Dist = np.vstack((Dist,np.array([j, i, d])))



Dist = np.round(Dist, 4)
```

10. Sort Dist matrix based on last column.

In [138…

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

In [139…

```
print(Dist)
```

```
[[   2.          9.         28.3019]
 [   4.          4.         35.    ]
 [   3.          0.         82.201 ]
 [   2.          1.         84.119 ]
 [   4.          1.        114.0044]
 [   1.          2.        117.1068]
 [   6.          0.        130.4952]
 [   3.          6.        135.0926]
 [   8.          5.        139.3879]
 [   4.          9.        142.3517]
 [   0.          3.        144.3468]
 [   4.          0.        151.0132]
 [   0.          1.        156.7323]
 [   0.          9.        165.5838]
 [   6.          6.        167.547 ]
 [   3.          1.        174.8742]
 [   6.          1.        182.2224]
 [   0.          0.        198.1237]
 [   2.          4.        202.0544]
 [   1.          9.        203.578 ]
 [   2.          0.        205.548 ]
 [   7.          5.        206.5188]
 [   5.          9.        211.7664]
 [   6.          3.        222.4275]
 [   3.          4.        226.7708]
 [   6.          9.        226.9273]
 [   5.          4.        227.1079]
 [   3.          9.        230.3606]
 [   1.          3.        241.9628]
 [   5.          1.        245.2835]
 [   9.          3.        251.0558]
 [   1.          1.        252.6599]
 [   0.          2.        258.0698]
 [   6.          4.        267.479 ]
 [   1.          8.        280.1303]
 [   2.          2.        282.0869]
 [   2.          3.        282.795 ]
 [   3.          3.        285.3279]
 [   0.          4.        288.0851]
 [   0.          6.        295.1779]
 [   9.          8.        317.72  ]
 [   8.          3.        338.1863]
 [   4.          6.        340.7873]
 [   9.          5.        341.1774]
 [   5.          0.        341.6577]
 [   9.          2.        344.3617]
 [   7.          7.        351.4669]
 [   1.          0.        360.2569]
 [   1.          4.        377.1538]
 [   2.          6.        379.0211]
 [   6.          2.        383.8359]
 [   8.          7.        398.1809]
 [   6.          5.        400.2512]
 [   0.          8.        408.6673]
 [   4.          3.        413.4562]
 [   3.          2.        434.0046]
 [   3.          5.        441.4533]
 [   0.          5.        445.4582]
 [   5.          2.        445.6467]
 [   2.          8.        449.0212]
 [   4.          2.        452.2322]
 [   8.          6.        492.1991]
 [   7.          3.        496.2308]
 [   1.          6.        499.0751]
 [   5.          3.        502.2519]
 [   8.          2.        518.3763]
```

```
 [   6.          8.        527.3111]
 [   8.          8.        533.6113]
 [   5.          6.        535.4521]
 [   9.          9.        545.7444]
 [   9.          1.        551.088 ]
 [   9.          6.        563.7065]
 [   9.          0.        574.5433]
 [   8.          0.        578.5404]
 [   3.          8.        583.3978]
 [   6.          7.        585.4912]
 [   2.          5.        600.4965]
 [   5.          8.        601.2021]
 [   3.          7.        601.7259]
 [   8.          1.        602.8806]
 [   7.          6.        604.3683]
 [   1.          5.        606.5056]
 [   9.          7.        617.3629]
 [   4.          8.        619.1551]
 [   8.          9.        620.1967]
 [   4.          5.        664.3561]
 [   0.          7.        670.7682]
 [   7.          8.        671.6621]
 [   7.          2.        672.4232]
 [   9.          4.        683.172 ]
 [   8.          4.        713.6281]
 [   7.          0.        717.2419]
 [   7.          1.        753.0485]
 [   7.          9.        774.0317]
 [   2.          7.        814.1775]
 [   5.          5.        816.1005]
 [   4.          7.        831.2015]
 [   7.          4.        856.8617]
 [   1.          7.        857.6456]
 [   5.          7.       1011.708 ]]
```

11. a. Get initial k rows from sorted matrix

In [140…

```
k = 10
M3 = Dist[:k]
print(M3)
```

```
[[  2.          9.         28.3019]
 [  4.          4.         35.    ]
 [  3.          0.         82.201 ]
 [  2.          1.         84.119 ]
 [  4.          1.        114.0044]
 [  1.          2.        117.1068]
 [  6.          0.        130.4952]
 [  3.          6.        135.0926]
 [  8.          5.        139.3879]
 [  4.          9.        142.3517]]
```

b. Get the rows corresponding to these k rows from M.

In [141…

```
M4 = np.empty((0, 3))
for rw in M3:
    M4 = np.vstack((M4,M[int(rw[0])]))
```

In [142…

```
M4
```

Out[142…

```
array([[196., 278.,   0.],
       [269., 124.,   0.],
       [440., 279.,   0.],
       [196., 278.,   0.],
       [269., 124.,   0.],
       [ 99., 424.,   0.],
       [421., 340.,   0.],
       [440., 279.,   0.],
       [561., 766.,   0.],
```

12. Find the number of 1s and number of 0s in the rows of M matrix, corresponding to the k rows found in Q. 11. Print 1 if the number of 1s are more else print 0.

In [143…

```
if(list(M4[:,2]).count(1.0) > list(M4[:,2]).count(0.0)):
    print("1")
else:
    print("0")
```

```
0
```

PART B: kNN implementation

a. Load diabetes dataset as done in Lab 1.

In [144…

```
path='diabetes.csv'
data=pd.read_csv(path)
```

b. Peek at few rows as done in Lab 1

In [145…

```
data
```

Out[145…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

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

In [146…

```
X = data.iloc[:,:8]
y = data.iloc[:,-1:]
```

In [147…

```
idx_80 = int(len(data) * 0.80)
X_train_1, X_test_1, y_train_1, y_test_1 = X[:idx_80], X[idx_80:],
y[:idx_80], y[idx_80:]
```

In [148…

```
X_train_1
```

Out[148…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
609	1	111	62	13	182	24.0	0.138	23
610	3	106	54	21	158	30.9	0.292	24
611	3	174	58	22	194	32.9	0.593	36
612	7	168	88	42	321	38.2	0.787	40
613	6	105	80	28	0	32.5	0.878	26

614 rows × 8 columns

d. Use inbuilt function to do splitting and interpret results

In [149…

```
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, shuffle=False)
```

In [150…

```
X_train
```

Out[150…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
609	1	111	62	13	182	24.0	0.138	23
610	3	106	54	21	158	30.9	0.292	24
611	3	174	58	22	194	32.9	0.593	36
612	7	168	88	42	321	38.2	0.787	40
613	6	105	80	28	0	32.5	0.878	26

614 rows × 8 columns

Interpretation: Both normal array slicing and train_test_split produced
same results because of shuffle=False property used in train_test_split function

e. Normalize the training dataset using StandardScaler as done in Lab 1.

Is it required to Normalize the testing dataset (X_test) too?

In [151…
```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

mnormalizer = preprocessing.Normalizer().fit(X_train)
X_train_norm = mnormalizer.transform(X_train)
X_test_norm = mnormalizer.transform(X_test)

std_scale = preprocessing.StandardScaler().fit(X_train)
X_train_std = std_scale.transform(X_train)
X_test_std  = std_scale.transform(X_test)
```

f. Invoke inbuilt kNN function.
Interpret the output obtained.

In [152…
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train.values.ravel())
y_pred = classifier.predict(X_test)
```

In [153…
```
y_pred
```

Out[153…
```
array([1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0],
      dtype=int64)
```

Interpretation: y_pred contains predictions derived from X_test on basis of model KNeighborsClassifier trained with train dataset X_train

g. Evaluate kNN.
Explain the output obtained.

In [154…
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[78 21]
 [25 30]]
              precision    recall  f1-score   support

           0       0.76      0.79      0.77        99
           1       0.59      0.55      0.57        55

    accuracy                           0.70       154
   macro avg       0.67      0.67      0.67       154
weighted avg       0.70      0.70      0.70       154
```

Interpretation: accuracy of the KNeighborsClassifier model for this dataset is 70%. Precision, recall and f1-score of predicting 0 is 0.76, 0.79 and 0.77 respectively. Precision, recall and f1-score of predicting 1 is 0.59, 0.55 and 0.57 respectively.

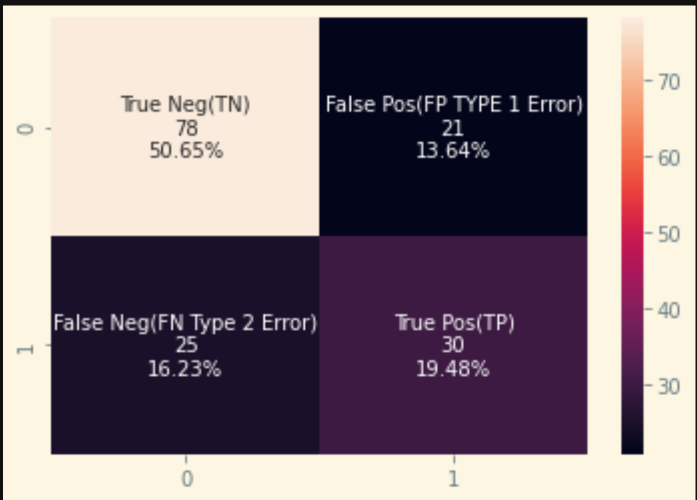
h. Find the total number of correct predictions, Type 1 error(FP) and Type 2 error(FN)

In [155…

```
from sklearn import metrics
cf_matrix = metrics.confusion_matrix(y_test, y_pred)
group_names = ['True Neg(TN)', 'False Pos(FP TYPE 1 Error)', 'False Neg(FN
Type 2 Error)', 'True Pos(TP)']
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
import seaborn as sns
sns.heatmap(cf_matrix, annot=labels, fmt='')
```

Out[155…

```
<AxesSubplot:>
```



i. Plot the Type 1 and Type 2 error for different values of 'k' values. (X-axis : k value and Y-axis : Type 1 error as well as Type 2 error).

In [156…

```
def KNN(k, X_train, y_train):
    classifier = KNeighborsClassifier(n_neighbors=k)
    classifier.fit(X_train, y_train.values.ravel())
    y_pred = classifier.predict(X_test)
    return y_pred
```

In [157…

```
y_pred_list = []
num_of_k = 30
for k in range(1, num_of_k):
    y_pred_list.append(KNN(k, X_train, y_train))
```

In [158…

```
cf_matrix[0][1], cf_matrix[1][0]
```

Out[158…

```
(21, 25)
```

In [159…

```
len(y_pred_list)
```

Out[159… 29

In [160…

```
fig, ax1 = plt.subplots()
x = list(range(1, num_of_k))
y1 = []
y2 = []
for k in range(0, num_of_k-1):
    cf_matrix = metrics.confusion_matrix(y_test, y_pred_list[k])
    y1.append(cf_matrix[0][1])
    y2.append(cf_matrix[1][0])
ax2 = ax1.twinx()
ax1.plot(x, y1, 'g-')
ax2.plot(x, y2, 'b-')

ax1.set_xlabel('X data')
ax1.set_ylabel('Y1 Type1 Error', color='g')
ax2.set_ylabel('Y2 Type2 Error', color='b')

plt.show()
```