In [1]:
```
'''
Q1
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

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


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

Out[1]:
```
2
```

In [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]]

'''
M=np.random.randint(1,101,size=(10,3))
print(M)
```

```
[[58 49 79]
 [99  4 97]
 [34 78 95]
 [32 81 59]
 [52 24 91]
 [37 63 24]
 [50  4 64]
 [48 90 56]
 [21 30 96]
 [29 17 90]]
```

In [3]:
```
# b. Print size of M.
M.shape
```

Out[3]:
```
(10, 3)
```

In [4]:
```
# c. Print only the number of rows of M
M.shape[0]
```

Out[4]:
```
10
```

In [5]:
```
#d. Print only the number of columns of M
M.shape[1]
```

Out[5]: 3

In [6]:
```
'''
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.
Example: The above matrix will change as
[[60 97 0]
[66 37 0]
…..
[64 64 1]]
'''

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

M
```

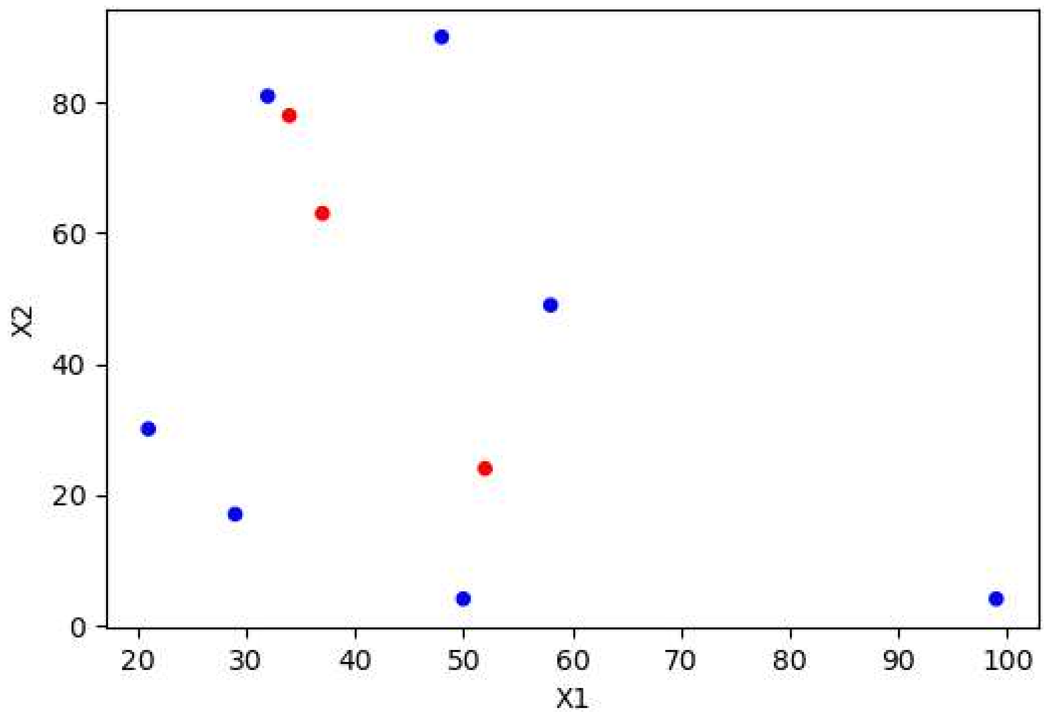
Out[6]:
```
array([[58, 49,  0],
       [99,  4,  0],
       [34, 78,  1],
       [32, 81,  0],
       [52, 24,  1],
       [37, 63,  1],
       [50,  4,  0],
       [48, 90,  0],
       [21, 30,  0],
       [29, 17,  0]])
```

In [7]:
```
#3. Create pandas dataframe ‘df’ from the created matrix M and name the columns as X1,
import pandas as pd
df=pd.DataFrame(M,columns=['X1','X2','Y'])
print(df)
```

```
   X1  X2  Y
0  58  49  0
1  99   4  0
2  34  78  1
3  32  81  0
4  52  24  1
5  37  63  1
6  50   4  0
7  48  90  0
8  21  30  0
9  29  17  0
```

In [8]:
```
# 4. Plot X1 and X2 using scatter plot. Color (X1, X2) red if the corresponding Y is 1

import matplotlib.pyplot as plt
col=df.Y.map({0:'b',1:'r'})
df.plot(kind='scatter',x='X1',y='X2',c=col,figsize=(6,4))
plt.show()
```



In [9]:
```
'''
Find the Squared error
    a. For two columns X1, X2, find squared error: (x1 – x2)^2
    Example: Matrix M will have [1369 841 ….. 0]
'''

squared_error=np.square(df['X1']-df['X2'])
print('Squared error :',squared_error)
```

```
Squared error : 0      81
1    9025
2    1936
3    2401
4     784
5     676
6    2116
7    1764
8      81
9     144
dtype: int32
```

In [10]:
```
# b. Find the sum of the squared error.

sum_squared_error = np.sum(squared_error)
print("Sum of Squared Error:", sum_squared_error)
```

```
Sum of Squared Error: 19008
```

In [11]:
```
#6. Find Euclidean distance between the first two rows of marix M.

euclidean_distance_n = np.sqrt(np.sum((M[0] - M[1])**2))
print("Euclidean Distance:", euclidean_distance_n)
```

```
Euclidean Distance: 60.876925020897694
```

In [12]:
```
euclidean_distance=np.linalg.norm(M[0]-M[1])
print("Euclidean Distance:", euclidean_distance)
```

```
Euclidean Distance: 60.876925020897694
```

In [13]:
```
#7. Create a vector V with three random values. Find the Euclidean distance between ea


V=np.random.randint(1,101,size=3)
print('Vector : ',V)

distances=np.linalg.norm(M-V,axis=1)
print('Euclidean distance between each row of M and V :\n',distances)
```

```
Vector :  [81 59 19]
Euclidean distance between each row of M and V :
 [31.46426545 60.90976933 53.79591063 56.97367813 48.88762625 47.70744177
 65.93178293 49.10193479 69.29646456 69.49100661]
```

In [14]:
```
'''
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.random.randint(1,51,size=(10,2))
print('A:\n',A)

# adding colmun

C=np.random.randint(1,51,size=10)
print('C:\n',C)
A=np.column_stack((A,C))
print('A:\n',A)

# adding row

R=np.random.randint(1,51,size=3)
print('R:\n',R)

A=np.vstack((A,R))
print('A:\n',A)
```

```
A:
 [[24 28]
 [ 7 26]
 [40  5]
 [31 13]
 [40 25]
 [26 22]
 [ 6 13]
 [37 11]
 [ 5 33]
 [12 22]]
C:
 [50 14 15 25 45 44  4 50 42  7]
A:
 [[24 28 50]
 [ 7 26 14]
 [40  5 15]
 [31 13 25]
 [40 25 45]
 [26 22 44]
 [ 6 13  4]
 [37 11 50]
 [ 5 33 42]
 [12 22  7]]
R:
 [16  6  8]
A:
 [[24 28 50]
 [ 7 26 14]
 [40  5 15]
 [31 13 25]
 [40 25 45]
 [26 22 44]
 [ 6 13  4]
 [37 11 50]
 [ 5 33 42]
 [12 22  7]
 [16  6  8]]
```

In [15]:

```
'''
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 sec
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(1,51,size=(10,2))
print('M1:\n',M1)
```

```
M1:
 [[46 25]
 [30 40]
 [ 7 21]
 [27 38]
 [17 10]
 [17 34]
 [21 49]
 [20 38]
 [ 3 41]
 [49 49]]
```

In [16]:

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

dist=euclidean_distances(M[:,:2],M1)
print(dist)
```

```
[[ 26.83281573  29.41088234  58.18075283  32.89376841  56.5862174
   43.65775991  37.          39.56008089  55.57877293   9.        ]
 [ 57.00877125  77.82673063  93.55746897  79.62411695  82.21921916
   87.31551981  90.04998612  86.00581376 102.88342918  67.26812024]
 [ 54.34151268  38.20994635  63.07138812  40.60788101  70.09279564
   47.16990566  31.78049716  42.3792402   48.27007354  32.64965543]
 [ 57.72347876  41.0487515   65.          43.28972164  72.56721023
   49.33558553  33.83784863  44.64302857  49.40647731  36.23534186]
 [  6.08276253  27.20294102  45.09988914  28.65309756  37.69615365
   36.40054945  39.8246155   34.92849839  51.86520992  25.17935662]
 [ 39.05124838  24.04163056  51.6139516   26.92582404  56.64803615
   35.22782991  21.26029163  30.23243292  40.49691346  18.43908891]
 [ 21.37755833  41.18252056  46.23851209  41.0487515   33.54101966
   44.59820624  53.53503526  45.3431362   59.81638571  45.01110974]
 [ 65.03076195  53.14132102  80.26207074  56.08029957  85.79627032
   64.00781202  49.09175083  59.05929224  66.52818951  41.01219331]
 [ 25.49509757  13.45362405  16.64331698  10.          20.39607805
    5.65685425  19.           8.06225775  21.09502311  33.83784863]
 [ 18.78829423  23.02172887  22.36067977  21.09502311  13.89244399
   20.80865205  32.984845    22.84731932  35.38361203  37.73592453]]
```

In [17]:

```python
Dist = []
for i in range(M.shape[0]):      # rows of M
    for j in range(M1.shape[0]): # rows of M1
        d = np.linalg.norm(M[i, :2] - M1[j])   # Euclidean distance
        Dist.append([i, j, d])    # row id of M, row id of M1, distance
Dist=np.array(Dist)
print(Dist[:10])
```

```
[[ 0.          0.         26.83281573]
 [ 0.          1.         29.41088234]
 [ 0.          2.         58.18075283]
 [ 0.          3.         32.89376841]
 [ 0.          4.         56.5862174 ]
 [ 0.          5.         43.65775991]
 [ 0.          6.         37.        ]
 [ 0.          7.         39.56008089]
 [ 0.          8.         55.57877293]
 [ 0.          9.          9.        ]]
```

In [18]:

```python
'''
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 whi
'''
```

```
sorted_Dist=Dist[Dist[:,-1].argsort()]
sorted_Dist[:10]
```

Out[18]:
```
array([[ 8.        ,  5.        ,  5.65685425],
       [ 4.        ,  0.        ,  6.08276253],
       [ 8.        ,  7.        ,  8.06225775],
       [ 0.        ,  9.        ,  9.        ],
       [ 8.        ,  3.        , 10.        ],
       [ 8.        ,  1.        , 13.45362405],
       [ 9.        ,  4.        , 13.89244399],
       [ 8.        ,  2.        , 16.64331698],
       [ 5.        ,  9.        , 18.43908891],
       [ 9.        ,  0.        , 18.78829423]])
```

In [19]:
```
#11. Get the initial k rows from the sorted matrix

k=11
k_rows=sorted_Dist[:k]
```

In [20]:
```
# 12. Find the number of 1s and 0s in the k rows above. Print 1 if the number of 1s is
ones=np.sum(k_rows==1)
zeros=np.sum(k_rows==0)

if ones> zeros:
    print(1)
else:
    print(0)
```

```
0
```

PART B : KNN Implementation

In [21]:
```
# a. Load diabetes dataset

df=pd.read_csv('diabetes_dataset.csv')
```

In [22]:
```
# b. Peek at a few rows.A
print(df.head())
```

```
   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
```

In [23]:
```
# c. Split the dataset into 80% training and 20% testing using numpy slicing

data=df.values
n=data.shape[0]
```

```
split_index=int(0.8*n)
train,test=data[:split_index],data[split_index:]
```

In [24]:
```
# d. Use the inbuilt function to do splitting and interpret results

from sklearn.model_selection import train_test_split
arr=data
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)
```

```
[[2.00e+00 8.80e+01 7.40e+01 ... 2.90e+01 2.29e-01 2.20e+01]
 [6.00e+00 1.15e+02 6.00e+01 ... 3.37e+01 2.45e-01 4.00e+01]
 [6.00e+00 1.02e+02 8.20e+01 ... 3.08e+01 1.80e-01 3.60e+01]
 ...
 [7.00e+00 1.29e+02 6.80e+01 ... 3.85e+01 4.39e-01 4.30e+01]
 [2.00e+00 1.14e+02 6.80e+01 ... 2.87e+01 9.20e-02 2.50e+01]
 [7.00e+00 1.33e+02 8.40e+01 ... 4.02e+01 6.96e-01 3.70e+01]]
```

In [25]:
```
# e. Do normalisation of training as well as testing dataset using StandardScaler

from sklearn.preprocessing import StandardScaler

scaler =StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

In [26]:
```
# f. Invoke inbuilt kNN function.

from sklearn.neighbors import KNeighborsClassifier

classifier=KNeighborsClassifier()
classifier.fit(X_train_scaled,y_train)
y_pred=classifier.predict(X_test_scaled)
```

In [27]:
```
# g. Evaluate KNN

from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

KNN_accuracy=accuracy_score(y_test,y_pred)
print("Accurace score : ",KNN_accuracy)
print('Confusion matrix :')
cm=confusion_matrix(y_test,y_pred)
print(cm)

print('\nClassification report :')
print(classification_report(y_test,y_pred))
```

```
Accurace score :  0.7272727272727273
Confusion matrix :
[[79 13]
 [29 33]]

Classification report :
               precision    recall  f1-score   support

         0.0       0.73      0.86      0.79        92
         1.0       0.72      0.53      0.61        62

    accuracy                           0.73       154
   macro avg       0.72      0.70      0.70       154
weighted avg       0.73      0.73      0.72       154
```

In [28]:

```python
# h. Find total number of correct predections

# Total correct predictions = TN + TP

total_correct_predections=cm[0,0]+cm[1,1]
print(total_correct_predections)
```

```
112
```

In [29]:

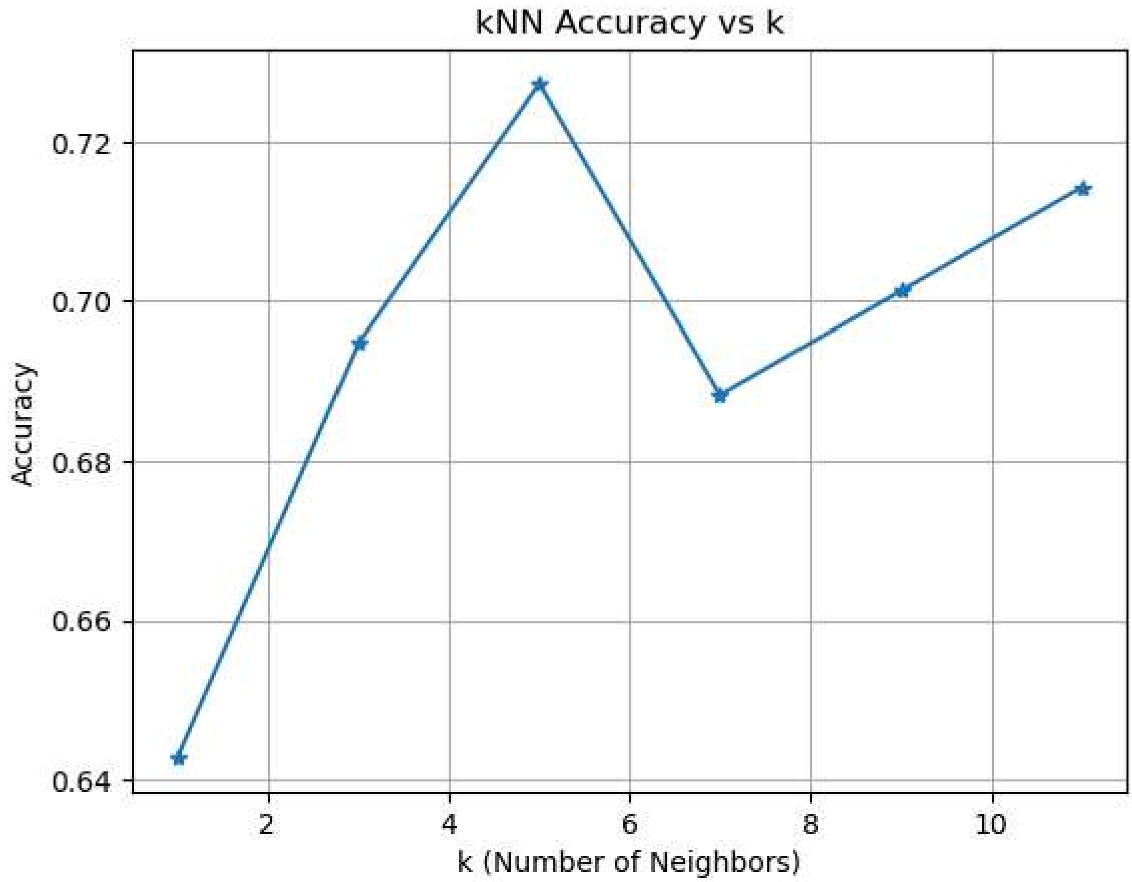
```python
# i. Repeat f, g, h for different values of k in kNN. And plot the graph

import matplotlib.pyplot as plt

k_values = [1, 3, 5, 7, 9, 11]
accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred_k = knn.predict(X_test_scaled)
    acc = (y_test == y_pred_k).sum() / len(y_test)
    accuracies.append(acc)

# Plot k vs accuracy
plt.plot(k_values, accuracies, marker='*')
plt.title("kNN Accuracy vs k")
plt.xlabel("k (Number of Neighbors)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```



In [30]:
```
# j. implement kNN on your own using the above exercises and apply them to the diabete

from collections import Counter
from sklearn.metrics import accuracy_score


def euclidean_distance(a,b):
    return np.sqrt(np.sum((a-b)**2))

def knn(X_train,y_train,X_test,k=n):
    y_pred=[]
    for test_point in X_test:

        distance=[euclidean_distance(test_point,x) for x in X_train]

        k_indices=np.argsort(distances)[:k]
        k_labels=[y_train[i] for i in k_indices]

        most_common =Counter(k_labels).most_common(1)[0][0]
        y_pred.append(most_common)

    return np.array(y_pred)
y_pred_manual = knn(X_train_scaled, y_train, X_test_scaled, k=5)
print(f"Manual KNN Accuracy: {accuracy_score(y_test, y_pred_manual):.2f}")
print(f"Python KNN Accuracy : {KNN_accuracy:.2f}",)
```

```
Manual KNN Accuracy: 0.40
Python KNN Accuracy : 0.73
```