In [2]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
import matplotlib.pyplot as plt
```

In [3]:
```
import pandas as pd
#imported the dataset given to us
pid=pd.read_csv("C:\\Users\\dhima\\anaconda3\\4rth week\\pima-indians-diabetes.cs

#made a copy of the original dataset
pid1=pid.copy()
print(pid1)
```

```
     pregs  plas  pres  skin  test   BMI   pedi  Age  class
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 x 9 columns]
```

In [4]:
```
#pid.columns
#created a list of all the columns present in the dataset
#pid_col=list(pid.columns)
pid2=pid.copy()
#pid_col
#pid_col1=pid_col.copy()
#we do not want to bring changes in the class column so we removed it from the co
#pid_col1.remove('class')
pid2.drop(["class"], axis = 1, inplace = True)

print(pid2)
```

```
     pregs  plas  pres  skin  test   BMI   pedi  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
..     ...   ...   ...   ...   ...   ...    ...  ...
763     10   101    76    48   180  32.9  0.171   63
764      2   122    70    27     0  36.8  0.340   27
765      5   121    72    23   112  26.2  0.245   30
766      1   126    60     0     0  30.1  0.349   47
767      1    93    70    31     0  30.4  0.315   23

[768 rows x 8 columns]
```

Ques 1 Show the performance of K-nearest neighbor (KNN) classifier for different values of K (1, 3, 5, 7, 9,

11, 13, 15, 17, 19, 21)

A. Find confusion matrix (use ‘confusion_matrix’) for each K.

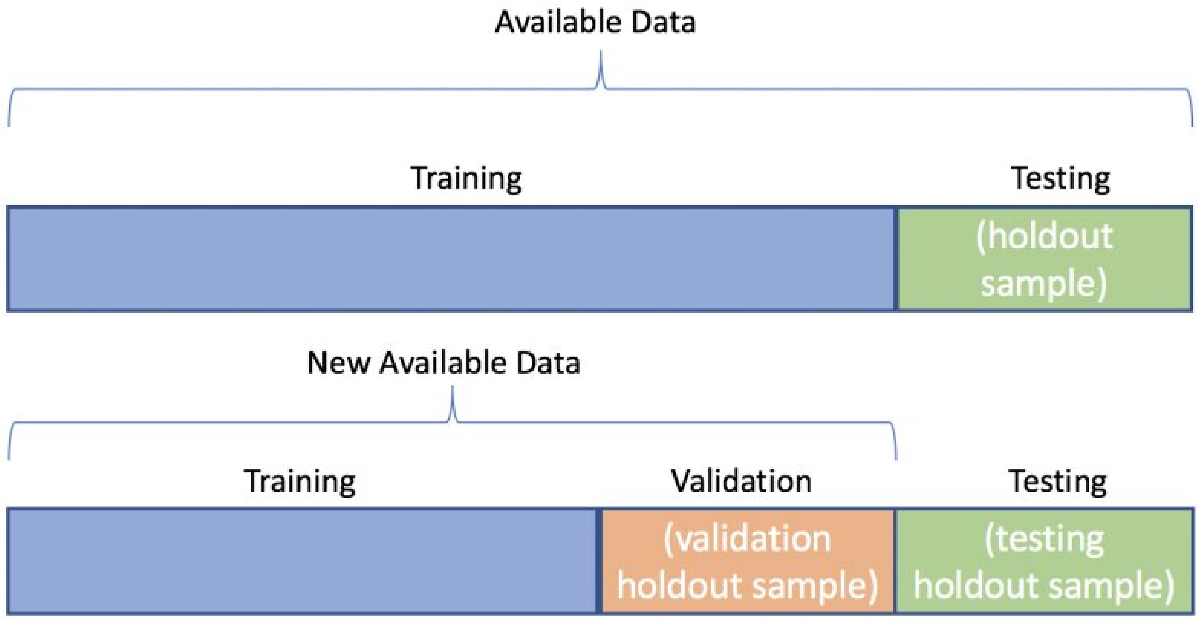
B. Find the classification accuracy (You can use ‘accuracy_score’) for each K. Note the value of K for

which the accuracy is high.

In [5]:
```
X1 = pid2                    # X denotes the input functions and here class defines
print(X1)
y1 = pid['class']                    #y denotes the output functions
print(y1)
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1,
                                                    train_size=0.7,
                                                    random_state=42,
                                                    stratify=y1)

print(X1_train)
print(X1_test)
print(y1_train)
print(y1_test)

print(f"Numbers of train instances by class: {np.bincount(y1_train)}")
print(f"Numbers of test instances by class: {np.bincount(y1_test)}")
```

```
     pregs  plas  pres  skin  test   BMI   pedi  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
..     ...   ...   ...   ...   ...   ...    ...  ...
763     10   101    76    48   180  32.9  0.171   63
764      2   122    70    27     0  36.8  0.340   27
765      5   121    72    23   112  26.2  0.245   30
766      1   126    60     0     0  30.1  0.349   47
767      1    93    70    31     0  30.4  0.315   23

[768 rows x 8 columns]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: class, Length: 768, dtype: int64
     pregs  plas  pres  skin  test   BMI   pedi  Age
209      7   184    84    33     0  35.5  0.355   41
176      6    85    78     0     0  31.2  0.382   42
147      2   106    64    35   119  30.5  1.400   34
454      2   100    54    28   105  37.8  0.498   24
636      5   104    74     0     0  28.8  0.153   48
..     ...   ...   ...   ...   ...   ...    ...  ...
214      9   112    82    32   175  34.2  0.260   36
113      4    76    62     0     0  34.0  0.391   25
556      1    97    70    40     0  38.1  0.218   30
759      6   190    92     0     0  35.5  0.278   66
107      4   144    58    28   140  29.5  0.287   37
```

```
[537 rows x 8 columns]
     pregs  plas  pres  skin  test   BMI   pedi  Age
730      3   130    78    23    79  28.4  0.323   34
198      4   109    64    44    99  34.8  0.905   26
24      11   143    94    33   146  36.6  0.254   51
417      4   144    82    32     0  38.5  0.554   37
387      8   105   100    36     0  43.3  0.239   45
..     ...   ...   ...   ...   ...   ...    ...  ...
94       2   142    82    18    64  24.7  0.761   21
437      5   147    75     0     0  29.9  0.434   28
86      13   106    72    54     0  36.6  0.178   45
221      2   158    90     0     0  31.6  0.805   66
19       1   115    70    30    96  34.6  0.529   32

[231 rows x 8 columns]
209    1
176    0
147    0
454    0
636    0
      ..
214    1
113    0
556    0
759    1
107    0
Name: class, Length: 537, dtype: int64
730    1
198    1
24     1
417    1
387    1
      ..
94     0
437    0
86     0
221    1
19     1
Name: class, Length: 231, dtype: int64
Numbers of train instances by class: [350 187]
Numbers of test instances by class: [150  81]
```


Available Data
Training
Testing
(holdout sample)
New Available Data
Training
Validation
Testing
(validation holdout sample)
(testing holdout sample)

In [6]:
```
X1_test, X1_val, y1_test, y1_val = train_test_split(X1_test, y1_test,
                                                    train_size=0.5,
                                                    random_state=42,
                                                    stratify=y1_test)
print(X1_train)
print(X1_val)
print(y1_test)
print(y1_val)

print(f"Numbers of test instances by class: {np.bincount(y1_test)}")
print(f"Numbers of validation instances by class: {np.bincount(y1_val)}")
```

```
     pregs  plas  pres  skin  test   BMI   pedi  Age
209      7   184    84    33     0  35.5  0.355   41
176      6    85    78     0     0  31.2  0.382   42
147      2   106    64    35   119  30.5  1.400   34
454      2   100    54    28   105  37.8  0.498   24
636      5   104    74     0     0  28.8  0.153   48
..     ...   ...   ...   ...   ...   ...    ...  ...
214      9   112    82    32   175  34.2  0.260   36
113      4    76    62     0     0  34.0  0.391   25
556      1    97    70    40     0  38.1  0.218   30
759      6   190    92     0     0  35.5  0.278   66
107      4   144    58    28   140  29.5  0.287   37

[537 rows x 8 columns]
     pregs  plas  pres  skin  test   BMI   pedi  Age
83       0   101    65    28     0  24.6  0.237   22
347      3   116     0     0     0  23.5  0.187   23
52       5    88    66    21    23  24.4  0.342   30
650      1    91    54    25   100  25.2  0.234   23
300      0   167     0     0     0  32.3  0.839   30
..     ...   ...   ...   ...   ...   ...    ...  ...
422      0   102    64    46    78  40.6  0.496   21
580      0   151    90    46     0  42.1  0.371   21
219      5   112    66     0     0  37.8  0.261   41
486      1   139    62    41   480  40.7  0.536   21
112      1    89    76    34    37  31.2  0.192   23

[116 rows x 8 columns]
418    0
235    1
373    0
330    0
64     1
      ..
151    0
535    1
82     0
25     1
637    0
Name: class, Length: 115, dtype: int64
83     0
347    0
52     0
650    0
300    1
```

```
       ..
422    0
580    1
219    1
486    0
112    0
Name: class, Length: 116, dtype: int64
Numbers of test instances by class: [75 40]
Numbers of validation instances by class: [75 41]
```

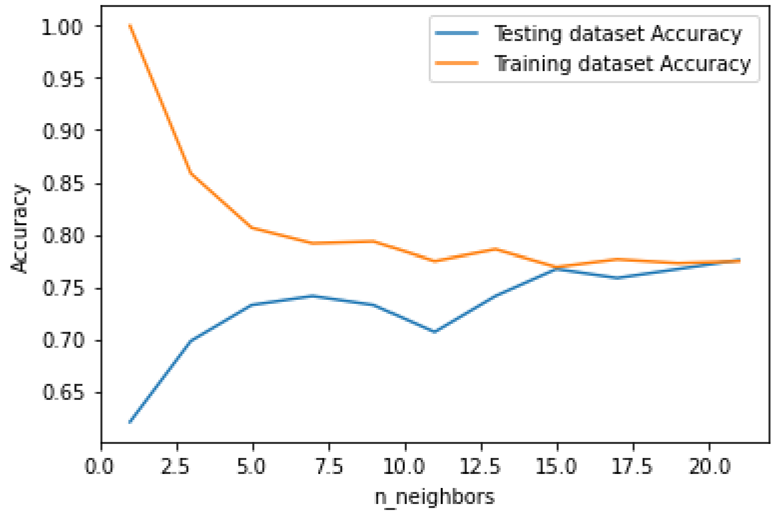
In [7]:
```
from sklearn.neighbors import KNeighborsClassifier

neighbors=[1,3,5,7,9,11,13,15,17,19,21]
train_accuracy = np.empty(len(neighbors))
val_accuracy = np.empty(len(neighbors))
acc=[]

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X1_train, y1_train)
    print('Predicted Outcomes for neighbours =',k,'are', knn.predict(X1_val))
    print('\n')

    print('Accuracy = ',knn.score(X1_val, y1_val))
    if ((knn.score(X1_val, y1_val))>=0):
        acc.append(knn.score(X1_val, y1_val))
    print('\n')

    matrix = confusion_matrix(y1_val,knn.predict(X1_val))
    print('Confusion Matrix = ',matrix)
    print('\n\n')


    # Compute traning and test data accuracy
    train_accuracy[i] = knn.score(X1_train, y1_train)
    val_accuracy[i] = knn.score(X1_val, y1_val)

print(acc)
print('\n')
print('maximum accuracy is =', max(acc)*100)


# Generate plot
plt.plot(neighbors, val_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')


plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

```
Predicted Outcomes for neighbours = 1 are [0 1 0 0 1 0 0 1 1 0 0 0 1 0 1 1 1 1
1 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 1 0 1 0
 0 0 1 1 0]


Accuracy =  0.6206896551724138


Confusion Matrix =  [[52 23]
 [21 20]]
```

```
Predicted Outcomes for neighbours = 3 are [0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 1 1 1
0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 1 0 1 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0
 1 0 1 1 0]


Accuracy =  0.6982758620689655


Confusion Matrix =  [[59 16]
 [19 22]]



Predicted Outcomes for neighbours = 5 are [0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0
0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0
 0 0 1 1 0]


Accuracy =  0.7327586206896551


Confusion Matrix =  [[64 11]
 [20 21]]



Predicted Outcomes for neighbours = 7 are [0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0
0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1
 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 0 1 1 0]


Accuracy =  0.7413793103448276


Confusion Matrix =  [[66  9]
 [21 20]]



Predicted Outcomes for neighbours = 9 are [0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0
0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1
 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 0 1 1 0]


Accuracy =  0.7327586206896551
```

```
Confusion Matrix =  [[64 11]
 [20 21]]



Predicted Outcomes for neighbours = 11 are [0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0
0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 0 0 1 0]


Accuracy =  0.7068965517241379


Confusion Matrix =  [[64 11]
 [23 18]]



Predicted Outcomes for neighbours = 13 are [0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 0
0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 1 0 1 0]


Accuracy =  0.7413793103448276


Confusion Matrix =  [[66  9]
 [21 20]]



Predicted Outcomes for neighbours = 15 are [0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 0
0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1
 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 1 0 1 0]


Accuracy =  0.7672413793103449


Confusion Matrix =  [[68  7]
 [20 21]]



Predicted Outcomes for neighbours = 17 are [0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0
0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1
 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 1 0 1 0]
```

```
Accuracy =  0.7586206896551724


Confusion Matrix =  [[67  8]
 [20 21]]



Predicted Outcomes for neighbours = 19 are [0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0
0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1
 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 1 0 1 0]


Accuracy =  0.7672413793103449


Confusion Matrix =  [[67  8]
 [19 22]]



Predicted Outcomes for neighbours = 21 are [0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0
0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1
 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 1 0 1 0]


Accuracy =  0.7758620689655172


Confusion Matrix =  [[68  7]
 [19 22]]



[0.6206896551724138, 0.6982758620689655, 0.7327586206896551, 0.741379310344827
6, 0.7327586206896551, 0.7068965517241379, 0.7413793103448276, 0.76724137931034
49, 0.7586206896551724, 0.7672413793103449, 0.7758620689655172]


maximum accuracy is = 77.58620689655173
```


Testing dataset Accuracy
Training dataset Accuracy
Accuracy
n_neighbors
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier

neighbors=[1,3,5,7,9,11,13,15,17,19,21]
#train_accuracy = np.empty(len(neighbors))
#val_accuracy = np.empty(len(neighbors))
acc=[]

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X1_train, y1_train)
    print('Predicted Outcomes for neighbours =',k,'are', knn.predict(X1_test))
    y_pred=knn.predict(X1_test)
    y_pred

    print('Accuracy = ',knn.score(X1_test, y1_test))
    if ((knn.score(X1_test, y1_test))>=0):
        acc.append(knn.score(X1_test, y1_test))
    print('\n')

    matrix = confusion_matrix(y1_test,knn.predict(X1_test))
    print('Confusion Matrix = ',matrix)

    print('\n')

print(acc)
print('\n')
print('maximum accuracy is =', max(acc)*100)
```

```
Predicted Outcomes for neighbours = 1 are [0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0
0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 1 0
 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7130434782608696


Confusion Matrix =  [[60 15]
 [18 22]]


Predicted Outcomes for neighbours = 3 are [0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7130434782608696


Confusion Matrix =  [[64 11]
 [22 18]]


Predicted Outcomes for neighbours = 5 are [0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1
 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 0
```

```
 1 0 0 0]
Accuracy =  0.7043478260869566


Confusion Matrix =  [[61 14]
 [20 20]]


Predicted Outcomes for neighbours = 7 are [0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0
0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 1 1 1
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7391304347826086


Confusion Matrix =  [[63 12]
 [18 22]]


Predicted Outcomes for neighbours = 9 are [0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1
1 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 1 1 1
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7130434782608696


Confusion Matrix =  [[61 14]
 [19 21]]


Predicted Outcomes for neighbours = 11 are [0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0
 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7043478260869566


Confusion Matrix =  [[62 13]
 [21 19]]


Predicted Outcomes for neighbours = 13 are [0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7217391304347827


Confusion Matrix =  [[63 12]
 [20 20]]


Predicted Outcomes for neighbours = 15 are [0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
```

```
1 0 1 0 1 0 1 0 0 0 0 1 1 0 0 0 1 0 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7130434782608696


Confusion Matrix =  [[63 12]
 [21 19]]


Predicted Outcomes for neighbours = 17 are [0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7130434782608696


Confusion Matrix =  [[64 11]
 [22 18]]


Predicted Outcomes for neighbours = 19 are [0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0
 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0
 1 0 0 0]
Accuracy =  0.7043478260869566


Confusion Matrix =  [[61 14]
 [20 20]]


Predicted Outcomes for neighbours = 21 are [0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 0
 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0
 0 0 0 0]
Accuracy =  0.7130434782608696


Confusion Matrix =  [[62 13]
 [20 20]]


[0.7130434782608696, 0.7130434782608696, 0.7043478260869566, 0.739130434782608
6, 0.7130434782608696, 0.7043478260869566, 0.7217391304347827, 0.71304347826086
96, 0.7130434782608696, 0.7043478260869566, 0.7130434782608696]


maximum accuracy is = 73.91304347826086
```

Ques.2 Build a Bayes classifier with Multi-modal Gaussian distribution (GMM) with Q components (modes)as class

conditional density for each class. Show the performance for different values of Q (2, 4, 8, 16).

Estimate the parameters of the Gaussian Mixture Model (mixture coefficients, mean vectors and covariance matrices) using maximum likelihood method.

A. Find confusion matrix (use ‘confusion_matrix’) for each Q.

B. Find the classification accuracy (You can use ‘accuracy_score’) for each Q.

C. Observe the values in the covariance matrix in each case and comment.

D. Compare the results with that obtained using Bayes classifier with unimodal Gaussian distribution

(Q = 1).

Clustering methods such as K-means have hard boundaries, meaning a data point either belongs to that cluster or it doesn't. On the other hand, clustering methods such as Gaussian Mixture Models (GMM) have soft boundaries, where data points can belong to multiple cluster at the same time but with different degrees of belief. e.g. a data point can have a 60% of belonging to cluster 1, 40% of belonging to cluster 2.

Apart from using it in the context of clustering, one other thing that GMM can be useful for is outlier detection: Due to the fact that we can compute the likelihood of each point being in each cluster, the points with a "relatively" low likelihood (where "relatively" is a threshold that we just determine ourselves) can be labeled as outliers.

This is the exact situation we're in when doing GMM. We have a bunch of data points, we suspect that they came from K different guassians, but we have no clue which data points came from which guassian. To solve this problem, we use the EM algorithm. The way it works is that it will start by placing guassians randomly (generate random mean and variance for the guassian). Then it will iterate over these two steps until it converges.

E step : With the current means and variances, it's going to figure out the probability of each data point xi coming from each guassian. M step : Once it computed these probability assignments it will use these numbers to re-estimate the guassians' mean and variance to better fit the data points.

That could be a problem for datasets with large number of dimensions (e.g. text data), because with the number of parameters growing roughly as the square of the dimension, it may quickly become impossible to find a sufficient amount of data to make good inferences. One common way to avoid this problem is to fix the covariance matrix of each component to be diagonal (off-diagonal value will be 0 and will not be updated). To achieve this, we can change the covariance_type parameter in scikit-learn's GMM to diag.

In [9]:
```
X2 = pid2                    # X denotes the input functions and here class defines
print(X2)
y2 = pid['class']                    #y denotes the output functions
print(y2)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2,
                                                     train_size=0.7,
                                                     random_state=42,
                                                     stratify=y2)

print(X2_train)
print(X2_test)
print(y2_train)
print(y2_test)
```

```
     pregs  plas  pres  skin  test   BMI   pedi  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
..     ...   ...   ...   ...   ...   ...    ...  ...
763     10   101    76    48   180  32.9  0.171   63
764      2   122    70    27     0  36.8  0.340   27
765      5   121    72    23   112  26.2  0.245   30
766      1   126    60     0     0  30.1  0.349   47
767      1    93    70    31     0  30.4  0.315   23

[768 rows x 8 columns]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: class, Length: 768, dtype: int64
     pregs  plas  pres  skin  test   BMI   pedi  Age
209      7   184    84    33     0  35.5  0.355   41
176      6    85    78     0     0  31.2  0.382   42
147      2   106    64    35   119  30.5  1.400   34
454      2   100    54    28   105  37.8  0.498   24
636      5   104    74     0     0  28.8  0.153   48
..     ...   ...   ...   ...   ...   ...    ...  ...
214      9   112    82    32   175  34.2  0.260   36
113      4    76    62     0     0  34.0  0.391   25
556      1    97    70    40     0  38.1  0.218   30
759      6   190    92     0     0  35.5  0.278   66
107      4   144    58    28   140  29.5  0.287   37

[537 rows x 8 columns]
     pregs  plas  pres  skin  test   BMI   pedi  Age
730      3   130    78    23    79  28.4  0.323   34
```

```
198      4  109   64   44   99  34.8  0.905   26
24      11  143   94   33  146  36.6  0.254   51
417      4  144   82   32    0  38.5  0.554   37
387      8  105  100   36    0  43.3  0.239   45
..     ...  ...  ...  ...  ...   ...    ...  ...
94       2  142   82   18   64  24.7  0.761   21
437      5  147   75    0    0  29.9  0.434   28
86      13  106   72   54    0  36.6  0.178   45
221      2  158   90    0    0  31.6  0.805   66
19       1  115   70   30   96  34.6  0.529   32

[231 rows x 8 columns]
209    1
176    0
147    0
454    0
636    0
      ..
214    1
113    0
556    0
759    1
107    0
Name: class, Length: 537, dtype: int64
730    1
198    1
24     1
417    1
387    1
      ..
94     0
437    0
86     0
221    1
19     1
Name: class, Length: 231, dtype: int64
```

In [10]:

```python
import numpy as np
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=2,covariance_type='full',max_iter=1000,n_init=2
clf=gm.fit(X2_train, y2_train)
y_pred=gm.predict(X2_test)       #NB classifier assumes that all the features are
print(y_pred)
```

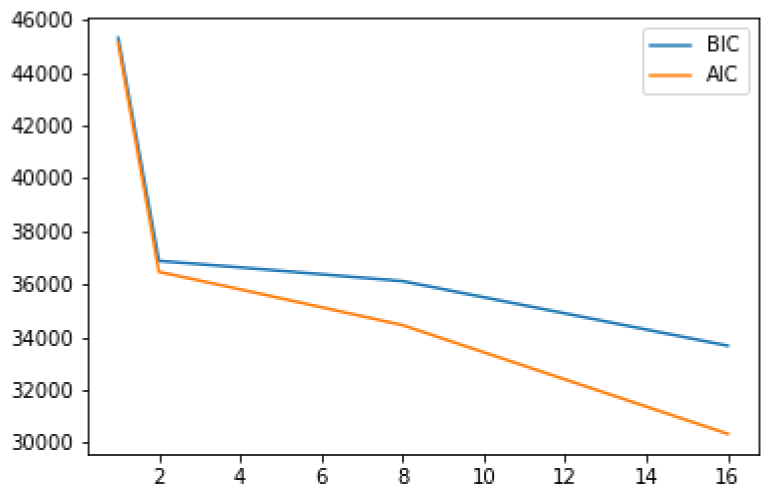
```
[0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 1 1 0 0 1 0 0 0
 0 0 1 0 1 1 0 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1 1 1 0 1 0 1 0 1 0 0 1 1 0 1 1
 1 1 0 0 1 1 0 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 0 0 0 0 1
 0 0 0 1 1 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0 0 0 1
 1 1 1 1 0 1 1 0 0 0 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 0 1 0
 1 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1
 0 1 0 0 0 1 1 1 0]
```

In [12]:
```
print(clf.bic(pid2))
print(clf.aic(pid2))
```

```
36949.88963138758
36536.59234513744
```

In [13]:
```
n_estimators=[1,2,4,8,16]
clfs=[GaussianMixture(n ).fit(pid2) for n in n_estimators]
bics=[clf.bic(pid2) for clf in clfs]
aics=[clf.aic(pid2) for clf in clfs]
print(clfs)
print(bics)
print(aics)
plt.plot(n_estimators, bics, label="BIC")    #low value of both aic and bic is pr
plt.plot(n_estimators, aics, label="AIC")
plt.legend();
```

```
[GaussianMixture(), GaussianMixture(n_components=2), GaussianMixture(n_componen
ts=4), GaussianMixture(n_components=8), GaussianMixture(n_components=16)]
[45311.82212435199, 36884.02231564019, 36637.73973732567, 36116.433283252394, 3
3669.879740016106]
[45107.49537609349, 36470.72502939005, 35806.50137509224, 34449.31276905238, 30
330.99492188293]
```



In [14]:
```
import numpy as np
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=16,covariance_type='full',max_iter=1000,n_init=
clf=gm.fit(X2_train, y2_train)
y_pred=gm.predict(X2_test)       #NB classifier assumes that all the features are
print(y_pred)

print('\n\n')
print(gm.means_)
print('\n\n')
print(gm.score(pid2, y=None))
print('\n\n')
probs = gm.predict_proba(pid2)
print(probs)
```

```
[ 1  1  0  9  9  1  8  9  1  8  1  5  1  9  6  9  5  1  1  9  6  8  8  1
 13  8  8  0  8  8  5 11  1  9 12 11 13  0 13  9  1  9  5  1  9 12  1  8
  8  9  1  1 13  9  9  1  7 13  8  8  9  1  8  7  8  1  8  1  1  9  9  0
  5  5  9  9  7  1  9  8  1  1  9  1  1  9  5 13 13  1 13  5  9  1  8  1
  0  8  8  8  8  7  8  9  9  9  1 11  1 13  8  0  1 11  8  9  1  8  7  8
  9  9  9  9  0  1  9  1 11  8  1  1  1  1  9  7  1  5  8  5  8  6  9  1
  7  1 13  8  8  8  8  9  1  8  8  7  0  1  1  9  7  8  6  8  5  1  8  8
 13  8  7  8  8  9  9  8  7  9  8  1  8  1 13  9 11  9  7  9  9  1  1  1
  1  7  8  9 12  9  1  1  1  5  8 13  0  8 11  1  8  1  6  7  9 13  8  8
  8  8  9  8  6  8  1  8  1 11  1  8  9  8  7]



[[3.16955352e+00 1.20855643e+02 6.85147670e+01 2.82706054e+01
  1.69410331e+02 3.35737721e+01 6.00045369e-01 3.16630470e+01]
 [2.56096203e+00 1.02477466e+02 6.85442609e+01 2.63139563e+01
  7.97271661e+01 3.10383569e+01 4.09326960e-01 2.66711606e+01]
 [5.20000000e+00 1.65600000e+02 7.24000000e+01 3.26000000e+01
  4.82600000e+02 3.35000000e+01 8.99000000e-01 3.90000000e+01]
 [1.17491023e+00 9.93445115e+01 6.93672324e+01 1.45426745e+01
  0.00000000e+00 2.35038960e+01 4.51625088e-01 2.38118098e+01]
 [7.00000000e+00 1.44000000e+02 8.10000000e+01 3.40000000e+01
  4.45000000e+01 2.87750000e+01 4.92750000e-01 5.35000000e+01]
 [4.45929422e+00 1.17714813e+02 9.94917866e-01 2.99682373e+00
  1.03637278e+00 2.54156772e+01 4.23465426e-01 3.23777928e+01]
 [3.80009513e+00 1.62905133e+02 7.59982751e+01 3.00003433e+01
  2.81800955e+02 3.67902014e+01 5.92580977e-01 4.05010866e+01]
 [1.29924246e+00 1.06500047e+02 6.76181878e+01 2.88263568e+01
  1.22479032e+02 3.42512948e+01 6.85807018e-01 2.80835182e+01]
 [4.97789633e+00 1.25499796e+02 7.51082292e+01 0.00000000e+00
  0.00000000e+00 3.15410869e+01 4.05552620e-01 3.78306967e+01]
 [4.26388802e+00 1.13186016e+02 7.35388708e+01 3.05317231e+01
  0.00000000e+00 3.34120599e+01 4.12666647e-01 3.50534934e+01]
 [4.00000000e+00 1.97000000e+02 7.00000000e+01 3.90000000e+01
  7.44000000e+02 3.67000000e+01 2.32900000e+00 3.10000000e+01]
 [5.70094030e+00 1.60061105e+02 7.52280570e+01 2.95928260e+01
  1.40629935e+02 3.27365471e+01 5.63938687e-01 3.76315845e+01]
 [6.29989779e+00 1.52200130e+02 7.40005862e+01 3.12997068e+01
  3.51992586e+02 3.39699132e+01 5.73080671e-01 3.90018065e+01]
 [4.58429871e+00 1.44162373e+02 7.55223564e+01 3.50779894e+01
  2.13550014e+02 3.62254968e+01 6.17786498e-01 3.84020991e+01]
```

```
 [2.00000000e+00 1.72000000e+02 7.24000000e+01 4.18000000e+01
  5.43400000e+02 3.87200000e+01 3.88200000e-01 3.32000000e+01]
 [3.25000000e+00 4.42500000e+01 7.60000000e+01 2.35000000e+01
  5.75000000e+00 3.04750000e+01 8.91750000e-01 2.95000000e+01]]
```

```
-20.868892014013216
```

```
[[1.86705755e-37 8.59182008e-14 0.00000000e+00 ... 1.03181708e-39
  0.00000000e+00 0.00000000e+00]
 [3.10636735e-43 1.75167068e-06 0.00000000e+00 ... 8.55580542e-46
  0.00000000e+00 0.00000000e+00]
 [2.53194702e-30 5.51066358e-23 0.00000000e+00 ... 8.65024706e-56
  0.00000000e+00 0.00000000e+00]
 ...
 [3.28194469e-04 9.99664053e-01 0.00000000e+00 ... 7.18147249e-13
  0.00000000e+00 0.00000000e+00]
 [1.30155539e-39 1.12438212e-20 0.00000000e+00 ... 2.30566179e-43
  0.00000000e+00 0.00000000e+00]
 [2.46643591e-41 9.46508124e-06 0.00000000e+00 ... 1.20813143e-50
  0.00000000e+00 0.00000000e+00]]
```