Question 1

In [1]:

```
#Q1
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:

```
#a
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pd.read_csv("iris.csv", names=names)
```

In [3]:

```
#b
dataset.shape
```

Out[3]:

```
(150, 5)
```

In [4]:

```
#c
dataset.groupby('class').size()
```

Out[4]:

```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

In [5]:

```
#d
# train / test dataset
array = dataset.values
X = array[:,0:4]
Y = array[:,4]
t_size = 0.20
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=t_size, random_state
=seed)
```

In [6]:

```
#e
# Make predictions
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_test)
print(accuracy_score(Y_test, predictions))
print(confusion_matrix(Y_test, predictions))
print(classification_report(Y_test, predictions))
```

```
0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
                precision    recall  f1-score   support
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11

       accuracy                           0.90        30
      macro avg       0.92      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30
```

In [7]:

```
#f
for i in range(1,11):
    # Make predictions
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, Y_train)
    predictions = knn.predict(X_test)
    print(accuracy_score(Y_test, predictions))
    print(confusion_matrix(Y_test, predictions))
    print(classification_report(Y_test, predictions))
```

```
0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11

       accuracy                           0.90        30
      macro avg       0.92      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30

0.9333333333333333
[[ 7  0  0]
 [ 0 12  0]
 [ 0  2  9]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.86      1.00      0.92        12
 Iris-virginica       1.00      0.82      0.90        11

       accuracy                           0.93        30
      macro avg       0.95      0.94      0.94        30
   weighted avg       0.94      0.93      0.93        30

0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11

       accuracy                           0.90        30
      macro avg       0.92      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30

0.9333333333333333
[[ 7  0  0]
 [ 0 12  0]
 [ 0  2  9]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.86      1.00      0.92        12
```

```
Iris-versicolor       0.86      1.00      0.92        12
 Iris-virginica       1.00      0.82      0.90        11

      accuracy                           0.93        30
     macro avg       0.95      0.94      0.94        30
  weighted avg       0.94      0.93      0.93        30

0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11

      accuracy                           0.90        30
     macro avg       0.92      0.91      0.91        30
  weighted avg       0.90      0.90      0.90        30

0.8666666666666667
[[ 7  0  0]
 [ 0 11  1]
 [ 0  3  8]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.79      0.92      0.85        12
 Iris-virginica       0.89      0.73      0.80        11

      accuracy                           0.87        30
     macro avg       0.89      0.88      0.88        30
  weighted avg       0.87      0.87      0.87        30

0.8666666666666667
[[ 7  0  0]
 [ 0 10  2]
 [ 0  2  9]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.83      0.83      0.83        12
 Iris-virginica       0.82      0.82      0.82        11

      accuracy                           0.87        30
     macro avg       0.88      0.88      0.88        30
  weighted avg       0.87      0.87      0.87        30

0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11

      accuracy                           0.90        30
     macro avg       0.92      0.91      0.91        30
  weighted avg       0.90      0.90      0.90        30

0.9
[[ 7  0  0]
 [ 0 10  2]
 [ 0  1 10]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.91      0.83      0.87        12
 Iris-virginica       0.83      0.91      0.87        11
```

```
       accuracy                           0.90        30
      macro avg       0.91      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30

0.9
[[ 7  0  0]
 [ 0 10  2]
 [ 0  1 10]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.91      0.83      0.87        12
 Iris-virginica       0.83      0.91      0.87        11

       accuracy                           0.90        30
      macro avg       0.91      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30
```

In [8]:

```
#g
for i in range(1,11):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=t_size, random_s
tate=i)
    knn = KNeighborsClassifier()
    knn.fit(X_train, Y_train)
    predictions = knn.predict(X_test)
    print(accuracy_score(Y_test, predictions))
```

```
1.0
1.0
0.9666666666666667
0.9666666666666667
0.9333333333333333
0.9666666666666667
0.9
0.9
1.0
0.9666666666666667
```

Question 2

In [61]:

```
#We do not need to import libraries as we have already imported them above for Question 1
#read data
data_train = pd.read_csv("Occupency_Detection/datatraining.txt")
data1_test =  pd.read_csv("Occupency_Detection/datatest.txt")
data2_test =  pd.read_csv("Occupency_Detection/datatest2.txt")

#drop date column
data_train = data_train.drop(['date'], axis=1)
data1_test = data1_test.drop(['date'], axis=1)
data2_test = data2_test.drop(['date'], axis=1)

#assign training and test data to variables so they become more readable and easy to use
#training set
T1_train_in = data_train.iloc[:, 0:5]
T1_train_out = data_train.iloc[:, 5]

#testing set
X1_test = data1_test.iloc[:, 0:5]
X2_test = data2_test.iloc[:, 0:5]
Y1_test = data1_test.iloc[:, 5]
Y2_test = data2_test.iloc[:, 5]

# print(data_train.shape)
# print(data1_test.shape)
```

```
# print(data2_test.shape)

# print(data_train.groupby('Occupancy').size())

#apply KNN
knn = KNeighborsClassifier()
knn.fit(T1_train_in, T1_train_out)

predictions = knn.predict(X1_test)
print(accuracy_score(Y1_test, predictions))
print(confusion_matrix(Y1_test, predictions))
print(classification_report(Y1_test, predictions))
```

```
0.9425891181988743
[[1645   48]
 [ 105  867]]
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1693
           1       0.95      0.89      0.92       972

    accuracy                           0.94      2665
   macro avg       0.94      0.93      0.94      2665
weighted avg       0.94      0.94      0.94      2665
```

In [62]:

```
predictions = knn.predict(X2_test)
print(accuracy_score(Y2_test, predictions))
print(confusion_matrix(Y2_test, predictions))
print(classification_report(Y2_test, predictions))
```

```
0.9621616078753076
[[7385  318]
 [  51 1998]]
              precision    recall  f1-score   support

           0       0.99      0.96      0.98      7703
           1       0.86      0.98      0.92      2049

    accuracy                           0.96      9752
   macro avg       0.93      0.97      0.95      9752
weighted avg       0.97      0.96      0.96      9752
```

Question 3

In [2]:

```
# k-nearest neighbors on the Iris Flowers Dataset
from random import seed
from random import randrange
from csv import reader
from math import sqrt

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
```

```
        row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores

# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    distance = 0.0
```

```
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

# kNN Algorithm
def k_nearest_neighbors(train, test, num_neighbors):
    predictions = list()
    for row in test:
        output = predict_classification(train, row, num_neighbors)
        predictions.append(output)
    return(predictions)

# Test the kNN on the Iris Flowers dataset
seed(1)
filename = 'iris.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# evaluate algorithm
n_folds = 5
num_neighbors = 5
scores = evaluate_algorithm(dataset, k_nearest_neighbors, n_folds, num_neighbors)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

```
Scores: [96.66666666666667, 96.66666666666667, 100.0, 90.0, 100.0]
Mean Accuracy: 96.667%
```

In [ ]: