```python
import numpy as np
import pandas as pd
from statistics import mode
pd.set_option("display.max_rows", None, "display.max_columns", None)


# load the dataset(sl = sepal length, sw = sepal width, pl = petal length, pw = petal width,class = Target)
iris = pd.read_csv('C:\\Users\\Authorized User1\\Desktop\\Data mining\\assignment 2\\iris.txt',
                 names=["sl","sw","pl","pw","class"])

# 0 = "Iris-setosa",1 = "Iris-versicolor",2 = "Iris-virginica"
iris['class_n'] = np.select([(iris['class'] == "Iris-setosa"),(iris['class'] == "Iris-versicolor"),(iris['class'] == "Iris-virginica")], [0,1,2], default="none")

print(iris.info())
print (iris)


#1. Make a function of a KNN classifier to perform classification based on the majority
# voting of K nearest neighbors. Use the Minkowski distance function you made in HW2 as the distance measure.

def minkowski(data_frame,array,r):
    col_name = ['sl', 'sw', 'pl', 'pw']
    b = {}
    for (w,i, j, k, l) in zip(data_frame.index,data_frame[col_name[0]], data_frame[col_name[1]], data_frame[col_name[2]], data_frame[col_name[3]]):
        b[w] = round(((((abs(i - array[0])) ** r) + ((abs(j - array[1])) ** r) + ((abs(k - array[2])) ** r) + ((abs(l - array[3])) ** r)) ** ( 1 / r)),2)
    a = dict(sorted(b.items(),key=lambda item: item[1],reverse=False))
    return (a)

def knn(frame,array,r,k):
    d = {}
    for i in array.index:
        c = (minkowski(frame,(list(array.drop('class_n',axis = 1).loc[i])),r))
        pairs = {k: c[k] for k in list(c)[:k]}
        d[i] = (classification(list(pairs.keys())))
    k = pd.DataFrame(list(d.items()),columns = ['index','predicted'])
    k = k.set_index('index')
    return (pd.concat([array, k], axis=1, join="inner"))

```

```
def classification(j):
    h = []
    for i in range(len(j)):
        if j[i] in range(50):
            h.append(0)
        elif j[i] in range(50,100):
            h.append(1)
        else:
            h.append(2)
    return(mode(h))

def accuracy_and_confusion(host):
    confusion_matrix = pd.crosstab(host['class_n'], host['predicted'], rownames=['Actual'], colnames=['Predicted'])
    #print ('the confusion matrix')
    #print(confusion_matrix)
    l = 0
    for (i,j) in zip(host['class_n'],host['predicted']):
        if i == j:
            l += 1
        else:
            pass
    print ('Accuracy score in %')
    print((l/len(host))*100)


#2. For the IRIS dataset, prepare a training dataset and a testing dataset for classification model training and testing.
# For each class, take the first 40 samples into the training dataset, and the remaining 10 samples into the testing dataset.
# The resulting training dataset has 120 samples and testing dataset has 30 samples.
df = iris.drop('class',axis = 1)

x_train1 = (df.iloc[0:40,:])
x_train2 = (df.iloc[50:90:,:])
x_train3 = (df.iloc[100:140,:])



x_test1 = (df.iloc[40:50,:])
x_test2 = (df.iloc[90:100:,:])
x_test3 = (df.iloc[140:150,:])

x_train = pd.concat([x_train1,x_train2,x_train3])
x_test = pd.concat([x_test1,x_test2,x_test3])

print (len(x_train))
print (len(x_test))


#3.1    For each KNN parameter setting, report classification
```

```
accuracy and the confusion matrix.

print ('FOR FULL DATASET TRAINING_SET 120 AND TESTING_SET 30')

for e in [3,5,7]:
    for t in [1, 2, 4]:
        x_train = pd.concat([x_train1,x_train2,x_train3])
        x_test = pd.concat([x_test1,x_test2,x_test3])
        final = (knn(x_train,x_test,t,e))
        final["class_n"] = pd.to_numeric(df["class_n"])
        print ('for r = %d and k = %d'%(t,e))
        accuracy_and_confusion(final)
        print('==========================')


#3.2    Calculate and report the classification accuracy for each class at each parameter setting.

print ('FOR IRIS-SETOSA DATASET TRAINING_SET 40 AND TESTING_SET 10')

for e in [3,5,7]:
    for t in [1, 2, 4]:
            x_train = (iris.iloc[0:40, :])
            x_test = (iris.iloc[40:50, :])
            final = (knn(x_train,x_test,t,e))
            final["class_n"] = pd.to_numeric(df["class_n"])
            print ('for r = %d and k = %d'%(t,e))
            accuracy_and_confusion(final)
            print('==========================')

print ('FOR IRIS-VERSICOLOR DATASET TRAINING_SET 40 AND TESTING_SET 10')

for e in [3,5,7]:
    for t in [1, 2, 4]:
            x_train = (iris.iloc[50:90, :])
            x_test = (iris.iloc[90:100, :])
            final = (knn(x_train,x_test,t,e))
            final["class_n"] = pd.to_numeric(df["class_n"])
            print ('for r = %d and k = %d'%(t,e))
            accuracy_and_confusion(final)
            print('==========================')

print ('FOR IRIS-VERGINICA DATASET TRAINING_SET 40 AND TESTING_SET 10')

for e in [3,5,7]:
    for t in [1, 2, 4]:
            x_train = (iris.iloc[100:140, :])
            x_test = (iris.iloc[140:150, :])
            final = (knn(x_train,x_test,t,e))
```

```
            final["class_n"] = pd.to_numeric(df["class_n"])
            print ('for r = %d and k = %d'%(t,e))
            accuracy_and_confusion(final)
            print('==========================')

#3.3    Assume we use the average accuracy of each class as the overall model performance measure,
# find the best parameter setting that generated the highest average accuracy for 3 classes.

def ggg():
    frame = pd.DataFrame(columns=['k', 'p','acc_score'])
    for q in [0,50,100]:
        x_train = iris.drop('class',axis=1).iloc[q:q+40,:]
        x_test = iris.drop('class', axis=1).iloc[q+40:q + 50, :]
        for e in [3, 5, 7]:
            for t in [1, 2, 4]:
                final = (knn(x_train, x_test, t, e))
                final["class_n"] = pd.to_numeric(df["class_n"])
                frame = frame.append({ "k": e, 'p': t, 'acc_score': (accuracy_and_confusion(final))},ignore_index=True)
    return(frame)

ggg()

#4. As shown in the plot below, a simple decision tree is constructed to classify two iris flowers: Versicolor and
# Virginica using two features of petal width and petal length. Assume the binary decision boundary on Petal Length is 4.8,
# and the decision boundary on Petal Width is 1.7. Make a function to implement this simple decision tree and use your function to
# classify the 100 iris samples of Versicolor and Virginica. Report the classification accuracy, sensitivity, and specificity.
# Here we define sensitivity = accuracy for class Versicolor, and specificity = accuracy of class Virginica.


df_c = iris.drop('class_n',axis = 1).iloc[100:150,:]

df_c['predicted'] = np.select([(df_c["pl"] <= 4.8) & (df_c["pw"] <= 1.7),
                              (df_c["pl"] > 4.8) & (df_c["pw"] > 1.7),
                              (df_c["pl"] >= 4.8) & (df_c["pw"] <= 1.7),
                              (df_c["pl"] <= 4.8) & (df_c["pw"] >= 1.7)], ["Iris-versicolor", "Iris-virginica","Iris-
```

```
virginica","Iris-virginica"],default='')
print (df_c)
confusion_matrix = pd.crosstab(df_c['class'], df_c['predicted'], rownames=['Actual'], colnames=['Predicted'])
print ('the confusion matrix')
print(confusion_matrix)
l = 0
for (i,j) in zip(df_c['class'],df_c['predicted']):
    if i == j:
        l += 1
    else:
        pass
print ('Accuracy score in %')
print((l/len(df_c))*100)

```