Question 1

Part A

Decision Tree Results

Dataset	Default	0%	25%	50%	75%
australian	56.52% (2)	81.16% (7)	86.96% (2)	56.52% (2)	20.77% (7)
labor	61.11% (2)	94.44% (7)	44.44% (7)	61.11% (12)	44.44% (12)
diabetes	66.23% (2)	67.10% (7)	64.07% (12)	66.23% (2)	35.50% (27)
ionosphere	66.04% (2)	86.79% (7)	82.08% (27)	71.70% (7)	18.87% (12)

Part B

(4) increase overfitting by increasing max_depth of the decision tree.

Part C

(2) yes, for 1/4 of the datasets.
From the chart, the ionosphere dataset is the only one which has been improved among these four.

Question 2

Part A

when 'n_neighbors' equals to 2,the accuracy score for training dataset and test dataset is 0.8969404186795491 and 0.7681159420289855 respectively.
The picture below is what I got form my python console,

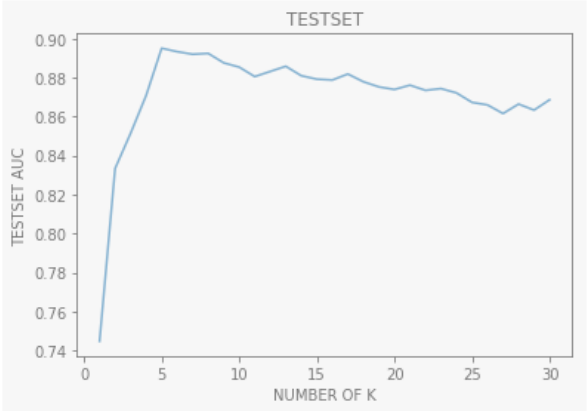
```
training accuacy score:0.8969404186795491
test accuacy score:0.7681159420289855
```

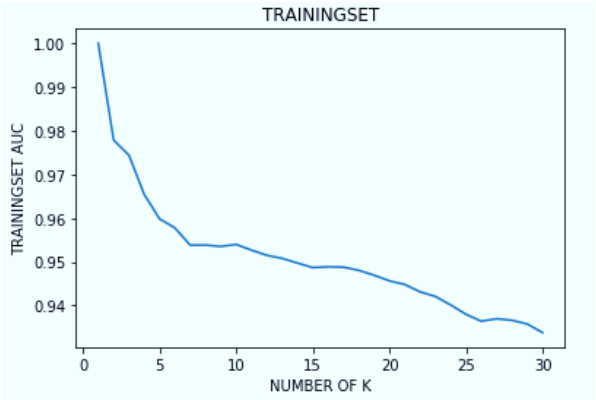
Part B

The optimal number of neighbours is 5.
I compared the value of AUC score of k (between 1 and 30) and found that 0.8950617283950617 is the biggest value when k is equal to 5.

Part C

Picture below is the test set and training set respectively





Part D

We have to compare the Precision score and recall score between k = 2 and k = 5,and the answer as follow,

	K=2	K=5
Precision score	0.7894736842105263	0.7666666666666667
Recall score	0.5555555555555556	0.8518518518518519

CODING PART

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 24 10:11:02 2019

@author: kyrie
"""
import numpy as np
with open("CreditCards.csv") as file:
    content = []
    x = []
    y = []
    x1 = []
    x2 = []
    x3 = []
    x4 = []
    x5 = []
    x6 = []
    x7 = []
    x8 = []
    x9 = []
    x10 = []
    x11 = []
    x12 = []
    x13 = []
    x14 = []
    for i in file:
        i = i.strip()
        content.append(i.split(','))
    content = content[1:]

    for i in content:
        x1.append(i[0])
        x2.append(i[1])
        x3.append(i[2])
        x4.append(i[3])
        x5.append(i[4])
        x6.append(i[5])
        x7.append(i[6])
        x8.append(i[7])
        x9.append(i[8])
        x10.append(i[9])
        x11.append(i[10])
        x12.append(i[11])
```

```
            x13.append(i[12])
            x14.append(i[13])
            y.append(i[14])

        x.append(x1)
        x.append(x2)
        x.append(x3)
        x.append(x4)
        x.append(x5)
        x.append(x6)
        x.append(x7)
        x.append(x8)
        x.append(x9)
        x.append(x10)
        x.append(x11)
        x.append(x12)
        x.append(x13)
        x.append(x14)

def pre_processing(x):
        a = []
        change = []
        for i in x:
            change.append(float(i))

        for i in change:
            x_new = (float(i)-min(change))/(max(change)-min(change))
            a.append(x_new)
        return a

##new:each data after normalization the last is the accurate outcome
new= []
for i in x:
        new.append(pre_processing(i))

###convert list
def transpose(matrix):
        new_matrix = []
        for i in range(len(matrix[0])):
            matrix1 = []
            for j in range(len(matrix)):
                matrix1.append(matrix[j][i])
            new_matrix.append(matrix1)
        return new_matrix
```

```
dataset = []
dataset =transpose(new)
####spliting the data part
training_set = dataset[:621]
training_outcome =pre_processing(y)[:621]
test_set = dataset[621:]
test_set_outcome =pre_processing(y)[621:]

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
auc_counter_training = []
auc_counter_testing = []


for i in range(1,31):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(training_set, training_outcome)
    test_set_predict = neigh.predict(test_set)
    train_set_predict = neigh.predict(training_set)

    train_score = accuracy_score(training_outcome,train_set_predict)
    test_score = accuracy_score(test_set_outcome,test_set_predict)
    test_set_predict_1 = neigh.predict_proba(test_set)[:,1]
    train_set_predict_1 = neigh.predict_proba(training_set)[:,1]
    #print(f'training accuacy score:{accuracy_score(training_outcome,train_set_predict)}')
    #print(f'test accuacy score:{accuracy_score(test_set_outcome,test_set_predict)}')
    auc_counter_training.append(roc_auc_score(training_outcome, train_set_predict_1))
    auc_counter_testing.append(roc_auc_score(test_set_outcome, test_set_predict_1))
#print("test auc",roc_auc_score(test_set_outcome, test_set_predict_1))
#print("training auc",roc_auc_score(training_outcome,train_set_predict_1))
#print(auc_counter_testing)


'''
#compare part
neigh = KNeighborsClassifier(n_neighbors=2)
neigh.fit(training_set, training_outcome)
test_set_predict = neigh.predict(test_set)
train_set_predict = neigh.predict(training_set)
```

```
train_score = accuracy_score(training_outcome,train_set_predict)
test_score = accuracy_score(test_set_outcome,test_set_predict)
test_set_predict_1 = neigh.predict_proba(test_set)[:,1]
train_set_predict_1 = neigh.predict_proba(training_set)[:,1]
print(f'training accuacy score:{accuracy_score(training_outcome,train_set_predict)}')
print(f'test accuacy score:{accuracy_score(test_set_outcome,test_set_predict)}')
auc_counter_training.append(roc_auc_score(training_outcome, train_set_predict_1))
auc_counter_testing.append(roc_auc_score(test_set_outcome, test_set_predict_1))
'''

import matplotlib.pyplot as plt

plt.plot([i for i in range(1,31)],auc_counter_testing)
plt.xlabel("NUMBER OF K")
plt.ylabel("TESTSET AUC")
plt.title("TESTSET")
plt.show()

plt.plot([i for i in range(1,31)],auc_counter_training)
plt.xlabel("NUMBER OF K")
plt.ylabel("TRAININGSET AUC")
plt.title("TRAININGSET")
plt.show()
'''
print(recall_score(test_set_outcome, test_set_predict))
print(precision_score(test_set_outcome, test_set_predict))

'''
```