Abhineet Raman(2002078)

B.sc.(H.) Computer Science  Semester: 6

Data Mining PracticAL File

**Index**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Topics** | **Page No.** |
| 1. | Index | 1 |
| 2. | Practical 1(Pre-processing) | 2-6 |
| 3. | Practical 2(Pre-processing) | 6-13 |
| 4. | Practical 3(Pre-processing) | 13-17 |
| 5. | Practical 4(Data Mining Techniques) | 17-19 |
| 6. | Practical 5(Data Mining Techniques) | 19-44 |
| 7. | Practical 6(Clustering) | 45-49 |
| 8. | Practical 7(Project) | 49-51 |

Q1. Create a file “people.txt” with the following data:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Age** | **agegroup** | **height** | **status** | **yearsmarried** |
| 21 | adult | 6.0 | single | -1 |
| 2 | child | 3 | married | 0 |
| 18 | adult | 5.7 | married | 20 |
| 221 | elderly | 5 | widowed | 2 |
| 34 | child | -7 | married | 3 |
|  |  |  |  |  |

i) Read the data from the file “people.txt”.

ii) Create a ruleset E that contain rules to check for the following conditions:

1. The age should be in the range 0-150.

2. The age should be greater than yearsmarried.

3. The status should be married or single or widowed.

4. If age is less than 18 the agegroup should be child, if age is between 18 and 65 the agegroup should be adult, if age is more than 65 the agegroup should be elderly.

iii) Check whether ruleset E is violated by the data in the file people.txt.

iv) Summarize the results obtained in part (iii)

v) Visualize the results obtained in part (iii)

Source Code:

people\_file = open('People.txt','r')

people\_data = [i.split() for i in (people\_file.readlines()[1::])]

# Create a ruleset E that contain rules to check for the following conditions:

# 1. The age should be in the range 0-150.

# 2. The age should be greater than yearsmarried.

# 3. The status should be married or single or widowed.

# 4. If age is less than 18 the agegroup should be child, if age is between 18 and 65 the agegroup

# should be adult, if age is more than 65 the agegroup should be elderly.

class People:

def \_\_init\_\_(self,age,ageGroup,height,status,yearMarried):

self.age = int(age)

self.ageGroup=ageGroup

self.height=float(height)

self.status=status

self.yearMarried=int(yearMarried)

def verifyAge(self):

if(self.age>=0 and self.age<=150):

return True

return False

def verifyYearsMarried(self):

if(self.age>self.yearMarried):

return True

return False

def verifyStatus(self):

check=['married','single','widowed']

if self.status.lower() in check:

return True

return False

def verifyAgeGroup(self):

if self.age<=18 and self.ageGroup.lower()=='child':

return True

elif self.age<=65 and self.ageGroup.lower()=='adult':

return True

elif self.age>65 and self.ageGroup.lower()=='elderly':

return True

else:

return False

def verify(self):

if self.verifyAge() and self.verifyAgeGroup() and self.verifyYearsMarried() and self.verifyStatus():

return True

return False

# Check whether ruleset E is violated by the data in the file people.txt.

peoples=[]

ageVisualize=[0,0]

ageGroupVisualize=[0,0]

statusVisualize=[0,0]

yearMarriedVisualize=[0,0]

for i in people\_data:

peoples.append(People(i[0],i[1],i[2],i[3],i[4]))

for i in range(0,len(peoples)):

print(f"People-{i+1}:-")

if peoples[i].verifyAge():

ageVisualize[0]+=1

else:

ageVisualize[1]+=1

print("Age: ","satisfied" if peoples[i].verifyAge() else "unsatisfied")

if peoples[i].verifyAgeGroup():

ageGroupVisualize[0]+=1

else:

ageGroupVisualize[1]+=1

print("AgeGroup: ","satisfied" if peoples[i].verifyAgeGroup() else "unsatisfied")

if peoples[i].verifyStatus():

statusVisualize[0]+=1

else:

statusVisualize[1]+=1

print("Status: ","satisfied" if peoples[i].verifyStatus() else "unsatisfied")

if peoples[i].verifyYearsMarried():

yearMarriedVisualize[0]+=1

else:

yearMarriedVisualize[1]+=1

print("Year-Married: ","satisfied" if peoples[i].verifyYearsMarried() else "unsatisfied")

print(ageVisualize)

print(ageGroupVisualize)

print(statusVisualize)

print(yearMarriedVisualize)

# Summarize the results obtained in above part

allVisualize=[0,0]

for i in range(0,len(peoples)):

if peoples[i].verify():

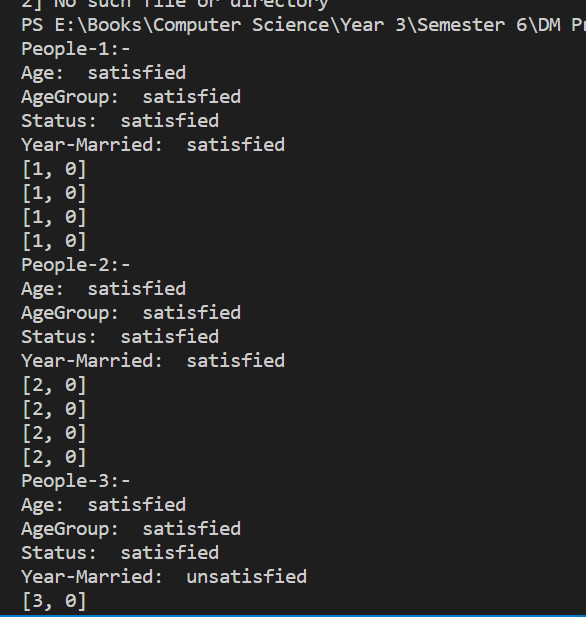
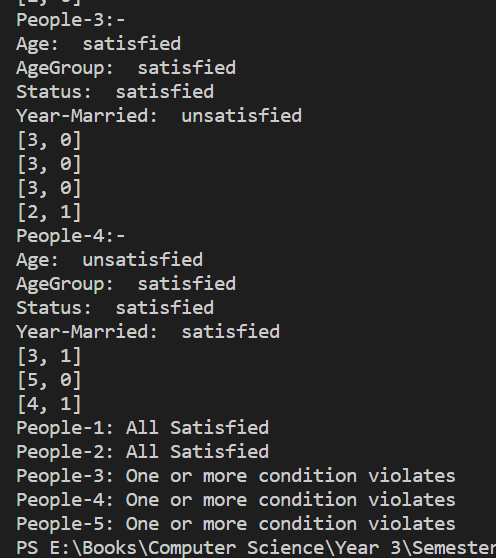
allVisualize[0]+=1

else:

allVisualize[1]+=1

print(f"People-{i+1}: {'All Satisfied' if peoples[i].verify() else 'One or more condition violates'}")

Output:

Q2. Perform the following preprocessing tasks on the dirty\_iris datasetii.

i) Calculate the number and percentage of observations that are complete.

ii) Replace all the special values in data with NA.

iii) Define these rules in a separate text file and read them.

(Use editfile function in R (package editrules). Use similar function in Python).

Print the resulting constraint object.

– Species should be one of the following values: setosa, versicolor or virginica.

– All measured numerical properties of an iris should be positive.

– The petal length of an iris is at least 2 times its petal width.

– The sepal length of an iris cannot exceed 30 cm.

– The sepals of an iris are longer than its petals.

iv) Determine how often each rule is broken (violatedEdits). Also summarize and plot the

result.

v) Find outliers in sepal length using boxplot and boxplot.stats

Source Code:

def isFloat(num):

    try:

        float(num)

        return True

    except ValueError:

        return False

class Iris:

    def \_\_init\_\_(self,sepalLength,sepalWidth,petalLength,petalWidth,species) -> None:

        self.sepalLength = float(sepalLength) if isFloat(sepalLength) else 0.0

        self.sepalWidth = float(sepalWidth) if isFloat(sepalWidth) else 0.0

        self.petalLength = float(petalLength) if isFloat(petalLength) else 0.0

        self.petalWidth = float(petalWidth) if isFloat(petalWidth) else 0.0

        self.species=species

    def checkSpecies(self):

        possibleValue = ["setosa","versicolor","virginica"]

        if (self.species in possibleValue):

            return True

        return False

    def checkPetalLengthSign(self):

        if (self.petalLength>0.0):

            return True

        return False

    def checkPetalWidthSign(self):

        if(self.petalWidth>0.0):

            return True

        return False

    def checkSepalLengthSign(self):

        if(self.sepalLength>0.0):

            return True

        return False

    def checkSepalWidthSign(self):

        if(self.sepalWidth>0.0):

            return True

        return False

    def checkPetalLength(self):

        if(self.petalLength>=2\*self.petalWidth):

            return True

        else:

            return False

    def checkSepalLength(self):

        if(self.sepalLength>0 and self.sepalLength<=30):

            return True

        return False

    def compareSepalPetal(self):

        if(self.sepalLength>self.petalLength):

            return True

        return False

import csv

# Calculate the number and percentage of observations that are complete.

complete\_count=0

Number\_of\_entries=0

rows=[]

with open('./iris.csv','r') as csvfile:

    iris\_data = csv.reader(csvfile)

    field = next(iris\_data)

    rows.append(field)

    for i in iris\_data:

        rows.append(i)

        Number\_of\_entries+=1

        if "NA" not in i:

            complete\_count+=1

complete\_percentage = (complete\_count/Number\_of\_entries)\*100

print("Complete Observations: ",complete\_count)

print("InComplete Observations: ",Number\_of\_entries-complete\_count)

print("Complete Observation Percentage: ",complete\_percentage,"%")

print("InComplete Observation Percentage: ",100-complete\_percentage,"%")

# print(rows)

# Replace all the special values in data with NA.

with open('./final\_iris.csv','w+',newline="",encoding='utf-8') as csvfile:

    writer = csv.writer(csvfile)

    writer.writerow(rows[0])

    for i in rows[1::]:

        if("Inf" in i):

            i[i.index("Inf")]="NA"

        writer.writerow(i)

# Define these rules in a separate text file and read them.

iris = []

with open('./final\_iris.csv','r') as csvfile:

    data = csv.reader(csvfile)

    header = next(data)

    for i in data:

        iris.append(Iris(\*i))

# Determine how often each rule is broken (violatedEdits). Also summarize and plot the result.

species=[0,0]

PLsign=[0,0]

PWsign=[0,0]

SLsign=[0,0]

SWsign=[0,0]

petalLength=[0,0]

sepalLength=[0,0]

sepalPetal=[0,0]

for i in iris:

    if(i.checkSpecies()):

        species[0]+=1

    else:

        species[1]+=1

    if(i.checkPetalLengthSign()):

        PLsign[0]+=1

    else:

        PLsign[1]+=1

        print(i.petalLength)

    if(i.checkPetalWidthSign()):

        PWsign[0]+=1

    else:

        PWsign[1]+=1

    if(i.checkSepalLengthSign()):

        SLsign[0]+=1

    else:

        SLsign[1]+=1

    if(i.checkSepalWidthSign()):

        SWsign[0]+=1

    else:

        SWsign[1]+=1

    if(i.checkPetalLength()):

        petalLength[0]+=1

    else:

        petalLength[1]+=1

    if(i.checkSepalLength()):

        sepalLength[0]+=1

    else:

        sepalLength[1]+=1

    if(i.compareSepalPetal()):

        sepalPetal[0]+=1

    else:

        sepalPetal[1]+=1

print("Valid iris on the basis of Species: ",species[0])

print("Invalid iris on the basis of Species: ",species[1])

print("Positive Petal Length: ",PLsign[0])

print("Negative Petal Length or NA: ",PLsign[1])

print("Positive Petal Width: ",PWsign[0])

print("Negative Petal Width or NA: ",PWsign[1])

print("Positive Sepal Length: ",SLsign[0])

print("Negative Sepal Length or NA: ",SLsign[1])

print("Positive Sepal Width: ",SWsign[0])

print("Negative Sepal Width or NA: ",SWsign[1])

print("Iris with Sepal Length less than 30cm: ",sepalLength[0])

print("Iris with Sepal Length more than 30cm: ",sepalLength[1])

print("Iris with its Petal Length equal to atleast two times of its Petal Width: ",petalLength[0])

print("Iris with its Petal Length equal to less than two times of its Petal Width: ",petalLength[1])

print("Iris with Sepal greater than its Petals: ",sepalPetal[0])

print("Iris with Sepal less than its Petals: ",sepalPetal[1])

# Visualization

# import matplotlib.pyplot as plt

# import numpy as np

# x=['Valid','Invalid']

# N=2

# width=0.10

# ind=np.arange(N)

# bar1=plt.bar(ind,species,width,color='red')

# bar2=plt.bar(ind+width,PLsign,width,color="green")

# bar3=plt.bar(ind+width\*2,PWsign,width,color="blue")

# bar4=plt.bar(ind+width\*3,SLsign,width,color="yellow")

# bar5=plt.bar(ind+width\*4,SWsign,width,color="orange")

# bar6=plt.bar(ind+width\*5,petalLength,width,color="skyblue")

# bar7=plt.bar(ind+width\*6,sepalLength,width,color="maroon")

# bar8=plt.bar(ind+width\*7,sepalPetal,width,color="purple")

# plt.ylabel("Number of Iris")

# plt.title("Overall Visualization")

# plt.xticks(ind+width,x)

# plt.legend((bar1,bar2,bar3,bar4,bar5,bar6,bar7,bar8),("Species","PetalLengthSign","PetalWidthSign","SepalLengthSign","SepalWidthSign","PetalLength","SepalLength","Relation between SepalLength and PetalLength"))

# plt.show()

# Find outliers in sepal length using boxplot and boxplot.stats

sepalLengthValues=[]

for i in iris:

    sepalLengthValues.append(i.sepalLength)

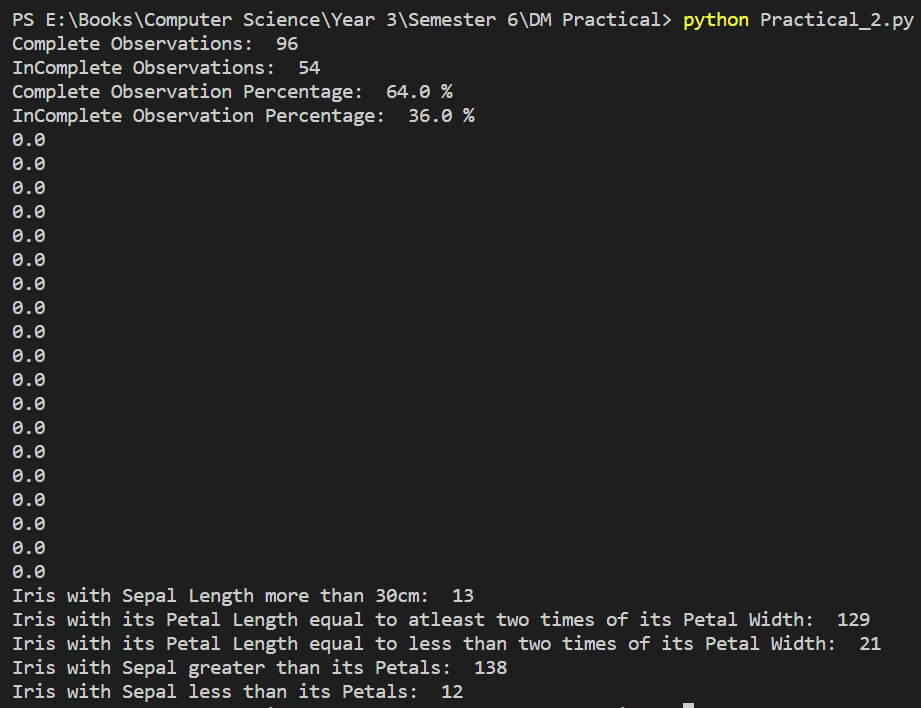
# print(sepalLengthValues)

# Sepal Length Visualization

# plt.boxplot(sepalLengthValues)

# plt.show()

Output:



Q3. Load the data from wine dataset. Check whether all attributes are standardized or not (mean is 0 and standard deviation is 1). If not, standardize the attributes. Do the same with Iris dataset.

Source Code:

# Load the data from wine dataset. Check whether all attributes are standardized or not (mean

# is 0 and standard deviation is 1). If not, standardize the attributes. Do the same with Iris dataset.

import statistics as st

import csv

meanWine=[0]

standardDeviationWine=[1]

with open("./wineDataset.csv",'r') as csvfile:

    wine\_data = csv.reader(csvfile)

    data\_Store = [[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

    for i in wine\_data:

        data\_Store[0].append(int(i[0]))

        for j in range(1,len(i)):

            data\_Store[j].append(float(i[j]))

    for i in range(1,len(data\_Store)):

        meanWine.append(st.mean(data\_Store[i]))

        standardDeviationWine.append(st.stdev(data\_Store[i]))

isMean=True

isStandardDeviation=True

for x in meanWine:

    if (round(x)!=0):

        isMean=False

for x in standardDeviationWine:

    if(round(x)!=1):

        isStandardDeviation=False

print(isMean)

print(isStandardDeviation)

if(isMean==False and isStandardDeviation==False):

    with open("./wineStandardDataset.csv",'w+',newline="",encoding='utf-8') as csvfile:

        writer = csv.writer(csvfile)

        for i in range(0,len(data\_Store[0])):

            tempRow=[]

            for j in range(0,len(data\_Store)):

                if j==0:

                    tempRow.append(data\_Store[j][i])

                else:

                    val=round((data\_Store[j][i]-meanWine[j])/standardDeviationWine[j],2)

                    tempRow.append(val)

            writer.writerow(tempRow)

# For Iris Data Set

def isfloat(num):

    if num == 'Inf':

        return False

    try:

        float(num)

        return True

    except ValueError:

        return False

meanIris=[]

standardDeviationIris=[]

data=[]

with open('./irisDataSet.csv','r') as csvfile:

    iris\_data = csv.reader(csvfile)

    field = next(iris\_data)

    data.append(field)

    iris\_data\_set=[[],[],[],[]]

    for i in iris\_data:

       data.append(i)

       if isfloat(i[0]):

        iris\_data\_set[0].append(float(i[0]))

       if isfloat(i[1]):

        iris\_data\_set[1].append(float(i[1]))

       if isfloat(i[2]):

        iris\_data\_set[2].append(float(i[2]))

       if isfloat(i[3]):

        iris\_data\_set[3].append(float(i[3]))

for i in iris\_data\_set:

    meanIris.append(round(st.mean(i),1))

    standardDeviationIris.append(round(st.stdev(i),1))

print(meanIris)

print(standardDeviationIris)

isMeanIris = True

isStdDeviationIris = True

for i in range(0,len(meanIris)):

    if round(meanIris[i])!=0:

        isMeanIris=False

    if round(standardDeviationIris[i])!=1:

        isStdDeviationIris=False

if (isMeanIris==False and isStdDeviationIris==False):

        with open("./irisDataSetStandard.csv",'w+',newline="",encoding='utf-8') as csvfile:

            writer = csv.writer(csvfile)

            writer.writerow(data[0])

            for i in data[1::]:

                temprow=[]

                for j in range(0,len(i)-1):

                    if isfloat(i[j]):

                        val = round((float(i[j])-meanIris[j])/standardDeviationIris[j],1)

                        temprow.append(val)

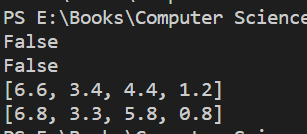
                    else:

                        temprow.append(i[j])

                temprow.append(i[4])

                writer.writerow(temprow)

Output:



Run following algorithms on 2 real datasets and use appropriate evaluation measures to compute correctness of obtained patterns:

Q4. Run Apriori algorithm to find frequent itemsets and association rules

1.1 Use minimum support as 50% and minimum confidence as 75%

1.2 Use minimum support as 60% and minimum confidence as 60 %

Source Code:

import pandas as pd

from mlxtend.frequent\_patterns import association\_rules

from mlxtend.frequent\_patterns import apriori

from mlxtend.preprocessing import TransactionEncoder

data = pd.read\_csv('./breast-cancer.csv')

dataset = [['A', 'B', 'C', 'D', 'F', 'H'],['B', 'E', 'F', 'H'],['A', 'C', 'E'],['B', 'C', 'D', 'F', 'H'],

['A', 'B', 'C', 'D', 'E'],['C','D','F','H'],['A','C','D','H'],['E','H']]

records=[]

for i in range(0,len(data)):

    records.append([str(data.values[i,j]) for j in range(0,10)])

TE = TransactionEncoder()

# For Breast cancer data

# TE\_ary = TE.fit(records).transform(records)

# For dataset

TE\_ary = TE.fit(dataset).transform(dataset)

df = pd.DataFrame(TE\_ary,columns=TE.columns\_)

print(df)

# Frequent Itemsets with minimum support 50%

frequent\_itemsets = apriori(df, min\_support=0.5, use\_colnames=True)

print(frequent\_itemsets)

# Association rules with minimum confidence 75%

print(association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.75))

# Frequent Itemsets with minimum support 60%

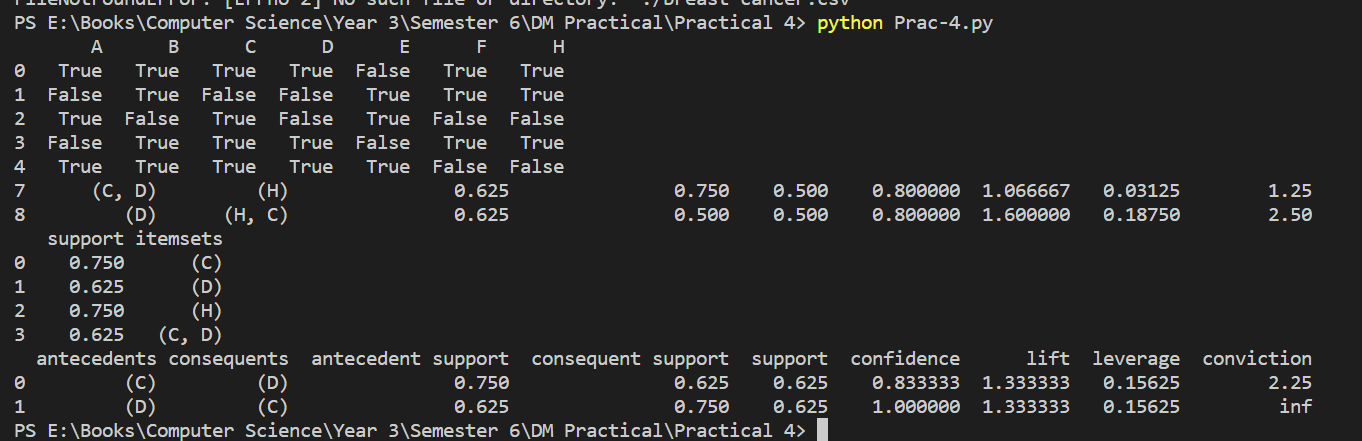
frequent\_itemsets = apriori(df, min\_support=0.6, use\_colnames=True)

print(frequent\_itemsets)

# Association rules with minimum confidence 60%

print(association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.6))

Output:



Q5. Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations:

5.1 a) Training set = 75% Test set = 25% b) Training set = 66.6% (2/3rd of total), Test set = 33.3%

5.2 Training set is chosen by i) hold out method ii) Random subsampling iii) Cross-Validation. Compare the accuracy of the classifiers obtained.

5.3 Data is scaled to standard format.

Source Code:

1.

# Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers.

# Divide the data set into training and test set. Compare the accuracy of the different classifiers

# under the following situations:

# 5.1 a) Training set = 75% Test set = 25% b) Training set = 66.6% (2/3rd of total), Test set =

# 33.3%

# 5.2 Training set is chosen by i) hold out method ii) Random subsampling iii) Cross-Validation.

# Compare the accuracy of the classifiers obtained.

# 5.3 Data is scaled to standard format.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import statistics

from sklearn.model\_selection import StratifiedKFold

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

def get\_Score(model,X\_train,Y\_train,X\_test,Y\_test):

    model.fit(X\_train,Y\_train)

    return model.score(X\_test,Y\_test)\*100

scale = StandardScaler()

data = pd.read\_csv('./abalone\_csv.csv')

X=data[0:].values[:,1:]

Y=data[0:].values[:,0]

Xval=["Hold-Out","Random Sub-Sampling","Cross-Validation"]

# Decision Tree Classifier

# 5.1

# Training Size:75% and Test Size:25%

print("Decision Tree Classifier")

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

clf=DecisionTreeClassifier()

clf.fit(X\_train,Y\_train)

print(f"Accuracy is {get\_Score(clf,X\_train,Y\_train,X\_test,Y\_test)} with 75% of training data")

# Training Size:66.6% and Test Size:33.3%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.333,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

clf.fit(X\_train,Y\_train)

print(f"Accuracy is {get\_Score(clf,X\_train,Y\_train,X\_test,Y\_test)} with 66.6% of training data")

# 5.2

# Using Hold-Out method for splitting

Accuracy = []

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

clf.fit(X\_train,Y\_train)

Accuracy.append(get\_Score(clf,X\_train,Y\_train,X\_test,Y\_test))

# using Random Subsampling for splitting

Accuracy\_Random=[]

k=6

for i in range(0,k):

    X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=100)

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    clf.fit(X\_train,Y\_train)

    prediction = clf.predict(X\_test)

    Accuracy\_Random.append(accuracy\_score(Y\_test,prediction)\*100)

Accuracy.append(statistics.mean(Accuracy\_Random))

# using K-Cross-Validation for splitting

k=9

kf = StratifiedKFold(n\_splits=k)

Accuracy\_kFold=[]

for train\_index,test\_index in kf.split(X,Y):

    X\_train,X\_test,Y\_train,Y\_test = X[train\_index],X[test\_index],Y[train\_index],Y[test\_index]

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_kFold.append(get\_Score(DecisionTreeClassifier(),X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_kFold))

print("Accuracy: ",Accuracy)

# Visualizing the accuracy of the Decision Tree Model for different Splitting models

Yval = Accuracy

plt.bar(Xval,Yval,color="green",width=0.2)

plt.xlabel("Splitting Method")

plt.title("Decision Tree Classifier Visualization")

plt.show()

# Naive Bayes Classifier

NBclf = GaussianNB()

print("Naive-Bayes Classifier")

# Training Size:75% and Test Size:25%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

print(f"Accuracy is {get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test)} with 75% of Training Data")

# Training Size:66.6% and Test Size:33.3%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.333,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

print(f"Accuracy is {get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test)} with 66.6% of Training Data")

# 5.2

# Using Hold-Out method for splitting

Accuracy = []

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000,shuffle=True,stratify=Y)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

Accuracy.append(get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test))

# using Random Subsampling for splitting

Accuracy\_Random=[]

k=6

for i in range(0,k):

    X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000,shuffle=True,stratify=Y)

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_Random.append(get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_Random))

# using K-Cross-Validation for splitting

k=9

kf = StratifiedKFold(n\_splits=k)

Accuracy\_kFold=[]

for train\_index,test\_index in kf.split(X,Y):

    X\_train,X\_test,Y\_train,Y\_test = X[train\_index],X[test\_index],Y[train\_index],Y[test\_index]

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_kFold.append(get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_kFold))

print("Accuracy: ",Accuracy)

# Visualizing the accuracy of the Naive-Bayes Classifier Model for different Splitting models

Yval = Accuracy

plt.bar(Xval,Yval,color="green",width=0.2)

plt.xlabel("Splitting Method")

plt.title("Naive Bayes Classifier Visualization")

plt.show()

# K-Nearest Neighbour Classifier

knn = KNeighborsClassifier(n\_neighbors=8)

print("K-Nearest Neighbour")

# Training Size:75% and Test Size:25%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

print(f"Accuracy is {get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test)} with 75% of Training Data")

# Training Size:66.6% and Test Size:33.3%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.333,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

print(f"Accuracy is {get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test)} with 66.6% of Training Data")

# 5.2

# Using Hold-Out method for splitting

Accuracy = []

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000,shuffle=True,stratify=Y)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

Accuracy.append(get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test))

# using Random Subsampling for splitting

Accuracy\_Random=[]

k=6

for i in range(0,k):

    X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000,shuffle=True,stratify=Y)

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_Random.append(get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_Random))

# using K-Cross-Validation for splitting

k=9

kf = StratifiedKFold(n\_splits=k)

Accuracy\_kFold=[]

for train\_index,test\_index in kf.split(X,Y):

    X\_train,X\_test,Y\_train,Y\_test = X[train\_index],X[test\_index],Y[train\_index],Y[test\_index]

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_kFold.append(get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_kFold))

print("Accuracy: ",Accuracy)

# Visualizing the accuracy of the K-Nearest Neighbour Model for different Splitting models

Yval = Accuracy

plt.bar(Xval,Yval,color="green",width=0.2)

plt.xlabel("Splitting Method")

plt.title("K-Nearest Neighbor Classifier Visualization")

plt.show()

2.

import pandas as pd

import math

import csv

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import statistics

from sklearn.model\_selection import StratifiedKFold

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

def get\_Score(model,X\_train,Y\_train,X\_test,Y\_test):

    model.fit(X\_train,Y\_train)

    return model.score(X\_test,Y\_test)\*100

scale = StandardScaler()

data = pd.read\_csv('./breast-cancer.csv')

updatedData=[[] for i in range(10)]

updatedData[0]=(list(data.values[:,0]))

X=data.values[:,1:]

i=0

for i in range(0,len(X)):

    index = X[i][0].index("-")

    lval = int(X[i][0][0:index])

    gval = int(X[i][0][index+1:])

    temp = lval + math.ceil((gval-lval)/2)

    updatedData[1].append(temp)

    if X[i][1] == "premeno":

        updatedData[2].append(0)

    elif X[i][1] == "ge40":

        updatedData[2].append(1)

    elif X[i][1] == "lt40":

        updatedData[2].append(2)

    else:

        updatedData[2].append(3)

    index = X[i][2].index("-")

    lval = int(X[i][2][0:index])

    gval = int(X[i][2][index+1:])

    temp = lval + math.ceil((gval-lval)/2)

    updatedData[3].append(temp)

    index = X[i][3].index("-")

    lval = int(X[i][3][0:index])

    gval = int(X[i][3][index+1:])

    temp = lval + math.ceil((gval-lval)/2)

    updatedData[4].append(temp)

    if X[i][4]=="no":

        updatedData[5].append(0)

    else:

        updatedData[5].append(1)

    updatedData[6].append(X[i][5])

    if X[i][6]=="left":

        updatedData[7].append(0)

    else:

        updatedData[7].append(1)

    if X[i][7]=="left\_low":

        updatedData[8].append(0)

    elif X[i][7]=="left\_up":

        updatedData[8].append(1)

    elif X[i][7]=="right\_low":

        updatedData[8].append(2)

    elif X[i][7]=="right\_up":

        updatedData[8].append(3)

    else:

        updatedData[8].append(4)

    if X[i][8]=="no":

        updatedData[9].append(0)

    else:

        updatedData[9].append(1)

with open("breast\_cancer\_updated.csv",'w',newline='') as file:

    writer = csv.writer(file)

    writer.writerow(["Class","Age","Menopause","Tumor-Size","Inv-nodes","Node-Caps","Deg-Malig","Breast","Breast-Quad","Irradiat"])

    for i in range(0,len(updatedData[0])):

        temp = [updatedData[j][i] for j in range(0,10)]

        print(temp)

        writer.writerow(temp)

def get\_Score(model,X\_train,Y\_train,X\_test,Y\_test):

    model.fit(X\_train,Y\_train)

    return model.score(X\_test,Y\_test)\*100

scale = StandardScaler()

newData = pd.read\_csv("./breast\_cancer\_updated.csv")

X=newData[0:].values[:,1:]

Y=newData[0:].values[:,0]

Xval=["Hold-Out","Random Sub-Sampling","Cross-Validation"]

# Decision Tree Classifier

# Training Size:75% and Test Size:25%

print("Decision-Tree Classifier")

clf=DecisionTreeClassifier()

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

clf.fit(X\_train,Y\_train)

print("Accuracy is {0:.3f} with 75% of training data".format(get\_Score(clf,X\_train,Y\_train,X\_test,Y\_test)))

# Training Size:66.6% and Test Size:33.3%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.333,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

clf=DecisionTreeClassifier()

clf.fit(X\_train,Y\_train)

print("Accuracy is {0:.3f} with 66.6% of training data".format(get\_Score(clf,X\_train,Y\_train,X\_test,Y\_test)))

# Using Hold-Out method for splitting

Accuracy = []

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

clf.fit(X\_train,Y\_train)

Accuracy.append(get\_Score(clf,X\_train,Y\_train,X\_test,Y\_test))

# using Random Subsampling for splitting

Accuracy\_Random=[]

k=6

for i in range(0,k):

    X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=100)

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    clf.fit(X\_train,Y\_train)

    prediction = clf.predict(X\_test)

    Accuracy\_Random.append(accuracy\_score(Y\_test,prediction)\*100)

Accuracy.append(statistics.mean(Accuracy\_Random))

# using K-Cross-Validation for splitting

k=9

kf = StratifiedKFold(n\_splits=k)

Accuracy\_kFold=[]

for train\_index,test\_index in kf.split(X,Y):

    X\_train,X\_test,Y\_train,Y\_test = X[train\_index],X[test\_index],Y[train\_index],Y[test\_index]

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_kFold.append(get\_Score(DecisionTreeClassifier(),X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_kFold))

print("Accuracy: ",Accuracy)

# Visualizing the accuracy of the Decision Tree Model for different Splitting models

Yval = Accuracy

plt.bar(Xval,Yval,color="green",width=0.2)

plt.xlabel("Splitting Method")

plt.title("Decision Tree Classifier Visualization")

plt.show()

# Naive Bayes Classifier

NBclf = GaussianNB()

print("Naive-Bayes Classifier")

# Training Size:75% and Test Size:25%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

print(f"Accuracy is {get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test)} with 75% of Training Data")

# Training Size:66.6% and Test Size:33.3%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.333,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

print(f"Accuracy is {get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test)} with 66.6% of Training Data")

# 5.2

# Using Hold-Out method for splitting

Accuracy = []

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000,shuffle=True,stratify=Y)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

Accuracy.append(get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test))

# using Random Subsampling for splitting

Accuracy\_Random=[]

k=6

for i in range(0,k):

    X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000,shuffle=True,stratify=Y)

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_Random.append(get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_Random))

# using K-Cross-Validation for splitting

k=9

kf = StratifiedKFold(n\_splits=k)

Accuracy\_kFold=[]

for train\_index,test\_index in kf.split(X,Y):

    X\_train,X\_test,Y\_train,Y\_test = X[train\_index],X[test\_index],Y[train\_index],Y[test\_index]

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_kFold.append(get\_Score(NBclf,X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_kFold))

print("Accuracy: ",Accuracy)

# Visualizing the accuracy of the Naive-Bayes Classifier Model for different Splitting models

Yval = Accuracy

plt.bar(Xval,Yval,color="green",width=0.2)

plt.xlabel("Splitting Method")

plt.title("Naive Bayes Classifier Visualization")

plt.show()

# K-Nearest Neighbour Classifier

knn = KNeighborsClassifier(n\_neighbors=8)

print("K-Nearest Neighbour")

# Training Size:75% and Test Size:25%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.25,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

print(f"Accuracy is {get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test)} with 75% of Training Data")

# Training Size:66.6% and Test Size:33.3%

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.333,random\_state=1000)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

print(f"Accuracy is {get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test)} with 66.6% of Training Data")

# 5.2

# Using Hold-Out method for splitting

Accuracy = []

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000,shuffle=True,stratify=Y)

X\_train = scale.fit\_transform(X\_train)

X\_test = scale.fit\_transform(X\_test)

Accuracy.append(get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test))

# using Random Subsampling for splitting

Accuracy\_Random=[]

k=6

for i in range(0,k):

    X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2,random\_state=1000,shuffle=True,stratify=Y)

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_Random.append(get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_Random))

# using K-Cross-Validation for splitting

k=9

kf = StratifiedKFold(n\_splits=k)

Accuracy\_kFold=[]

for train\_index,test\_index in kf.split(X,Y):

    X\_train,X\_test,Y\_train,Y\_test = X[train\_index],X[test\_index],Y[train\_index],Y[test\_index]

    X\_train = scale.fit\_transform(X\_train)

    X\_test = scale.fit\_transform(X\_test)

    Accuracy\_kFold.append(get\_Score(knn,X\_train,Y\_train,X\_test,Y\_test))

Accuracy.append(statistics.mean(Accuracy\_kFold))

print("Accuracy: ",Accuracy)

# Visualizing the accuracy of the K-Nearest Neighbour Model for different Splitting models

Yval = Accuracy

plt.bar(Xval,Yval,color="green",width=0.2)

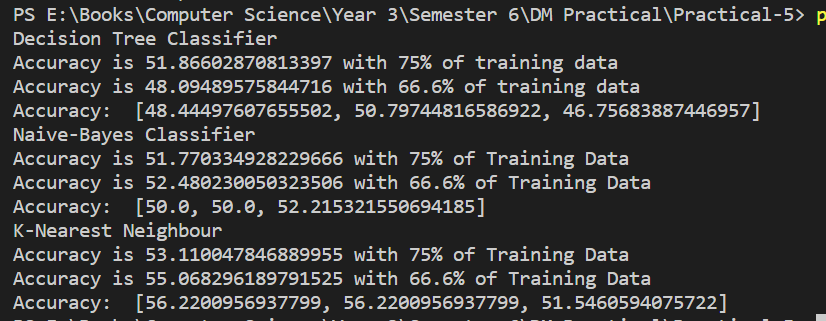
plt.xlabel("Splitting Method")

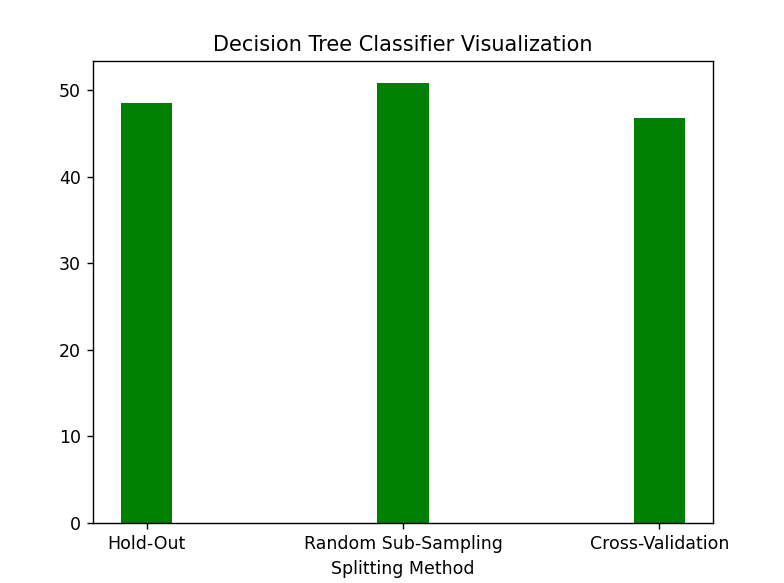
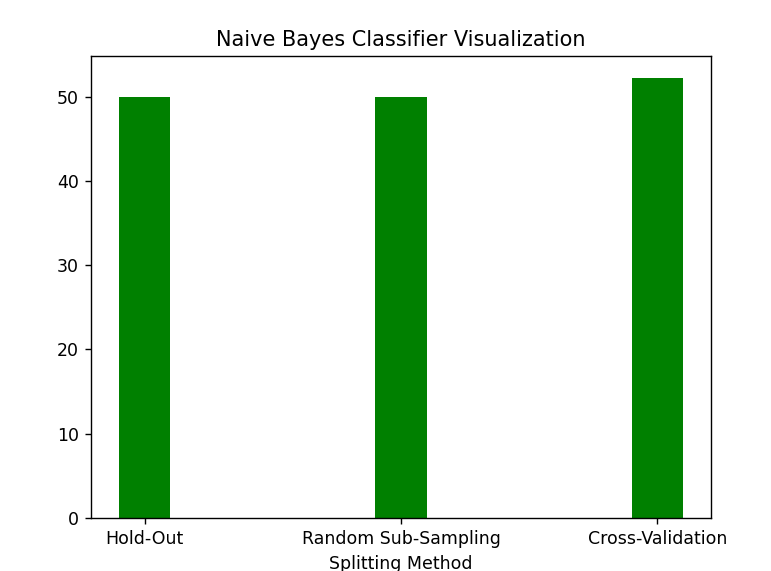
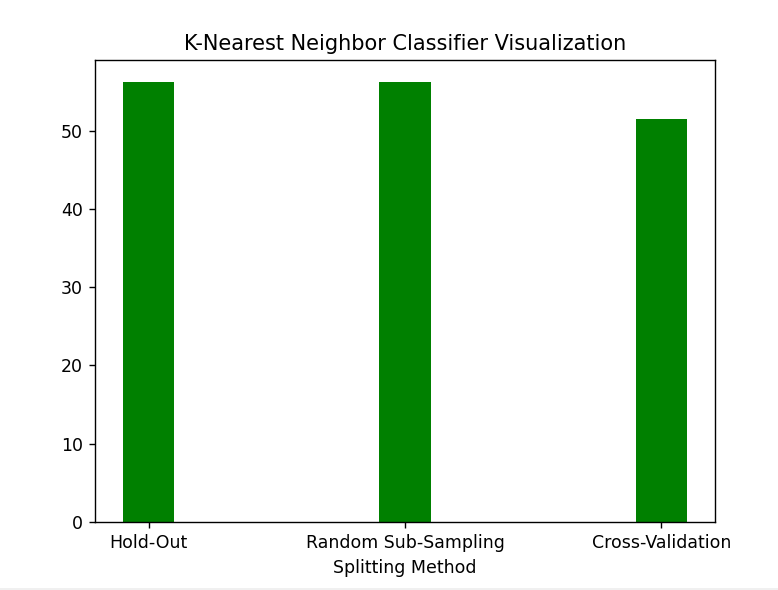
plt.title("K-Nearest Neighbor Classifier Visualization")

plt.show()

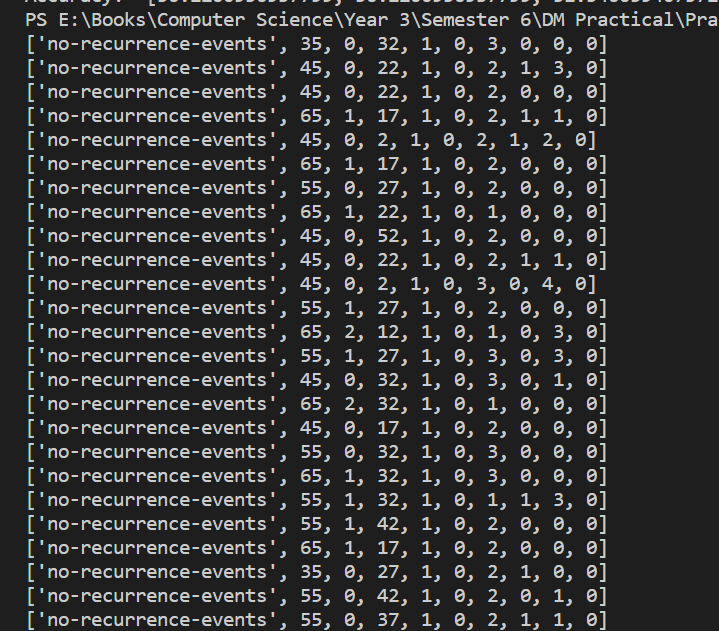
Output:

1.



2.



['no-recurrence-events', 65, 1, 17, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 35, 0, 27, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 55, 0, 42, 1, 0, 2, 0, 1, 0]

['no-recurrence-events', 55, 0, 37, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 45, 0, 27, 1, 0, 2, 0, 1, 0]

['no-recurrence-events', 55, 0, 22, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 65, 1, 27, 1, 0, 3, 1, 1, 0]

['no-recurrence-events', 45, 0, 42, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 65, 1, 32, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 55, 1, 42, 1, 0, 3, 1, 1, 0]

['no-recurrence-events', 55, 0, 17, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 55, 0, 12, 1, 0, 3, 0, 0, 0]

['no-recurrence-events', 55, 1, 12, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 55, 1, 12, 1, 0, 1, 0, 1, 0]

['no-recurrence-events', 35, 0, 32, 1, 0, 2, 0, 1, 0]

['no-recurrence-events', 55, 1, 2, 1, 0, 2, 0, 4, 0]

['no-recurrence-events', 55, 1, 17, 1, 0, 1, 1, 4, 0]

['no-recurrence-events', 45, 0, 12, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 45, 0, 32, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 55, 1, 22, 1, 0, 1, 1, 0, 0]

['no-recurrence-events', 65, 1, 27, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 65, 1, 7, 1, 0, 1, 0, 4, 0]

['no-recurrence-events', 45, 0, 12, 1, 0, 2, 0, 1, 0]

['no-recurrence-events', 55, 1, 52, 1, 0, 1, 1, 3, 0]

['no-recurrence-events', 55, 1, 32, 1, 0, 1, 0, 1, 0]

['no-recurrence-events', 45, 0, 27, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 55, 0, 27, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 45, 0, 22, 1, 0, 1, 1, 3, 0]

['no-recurrence-events', 45, 0, 22, 1, 0, 1, 1, 0, 0]

['no-recurrence-events', 55, 2, 17, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 35, 0, 22, 1, 0, 2, 0, 2, 0]

['no-recurrence-events', 55, 0, 17, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 75, 1, 22, 1, 0, 3, 0, 1, 0]

['no-recurrence-events', 75, 1, 42, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 75, 1, 42, 1, 0, 1, 1, 3, 0]

['no-recurrence-events', 55, 1, 2, 1, 0, 1, 1, 4, 0]

['no-recurrence-events', 55, 1, 7, 1, 0, 2, 1, 3, 0]

['no-recurrence-events', 65, 1, 32, 1, 0, 1, 0, 1, 0]

['no-recurrence-events', 65, 1, 17, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 45, 0, 22, 1, 0, 2, 0, 4, 0]

['no-recurrence-events', 45, 0, 12, 1, 0, 1, 1, 2, 0]

['no-recurrence-events', 55, 1, 2, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 25, 0, 37, 1, 0, 2, 1, 3, 0]

['no-recurrence-events', 45, 0, 27, 1, 0, 1, 0, 2, 0]

['no-recurrence-events', 45, 0, 12, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 45, 0, 27, 1, 0, 1, 1, 2, 0]

['no-recurrence-events', 55, 1, 22, 1, 0, 3, 0, 1, 0]

['no-recurrence-events', 55, 1, 37, 1, 0, 3, 0, 0, 0]

['no-recurrence-events', 65, 1, 52, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 65, 1, 12, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 45, 0, 27, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 65, 1, 22, 1, 0, 2, 0, 1, 0]

['no-recurrence-events', 55, 0, 17, 1, 0, 2, 1, 2, 0]

['no-recurrence-events', 35, 0, 7, 1, 0, 2, 0, 2, 0]

['no-recurrence-events', 55, 1, 12, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 55, 1, 12, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 35, 0, 27, 1, 0, 1, 0, 4, 0]

['no-recurrence-events', 55, 0, 27, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 45, 0, 27, 1, 0, 2, 1, 4, 0]

['no-recurrence-events', 55, 1, 12, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 65, 1, 12, 1, 0, 1, 0, 1, 0]

['no-recurrence-events', 65, 1, 17, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 55, 1, 17, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 45, 0, 22, 1, 0, 1, 0, 2, 0]

['no-recurrence-events', 55, 1, 37, 1, 0, 3, 0, 1, 0]

['no-recurrence-events', 65, 1, 27, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 75, 1, 2, 1, 0, 1, 0, 2, 0]

['no-recurrence-events', 55, 1, 22, 1, 0, 3, 1, 1, 0]

['no-recurrence-events', 45, 0, 42, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 35, 0, 2, 1, 0, 2, 1, 4, 0]

['no-recurrence-events', 55, 1, 22, 1, 0, 3, 0, 1, 0]

['no-recurrence-events', 55, 1, 27, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 65, 1, 22, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 55, 0, 12, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 45, 0, 32, 1, 0, 2, 1, 2, 0]

['no-recurrence-events', 65, 1, 32, 1, 0, 2, 0, 1, 0]

['no-recurrence-events', 65, 1, 17, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 45, 0, 32, 1, 0, 1, 0, 3, 0]

['no-recurrence-events', 35, 0, 27, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 45, 1, 22, 1, 0, 3, 0, 0, 0]

['no-recurrence-events', 55, 1, 32, 1, 0, 3, 1, 0, 0]

['no-recurrence-events', 55, 0, 27, 1, 0, 2, 1, 2, 0]

['no-recurrence-events', 45, 0, 22, 1, 0, 2, 0, 2, 0]

['no-recurrence-events', 45, 0, 12, 1, 0, 2, 1, 0, 0]

['no-recurrence-events', 45, 0, 32, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 45, 0, 22, 1, 0, 2, 0, 1, 0]

['no-recurrence-events', 35, 0, 42, 1, 0, 2, 1, 3, 0]

['no-recurrence-events', 45, 0, 32, 1, 0, 3, 1, 3, 0]

['no-recurrence-events', 65, 1, 32, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 55, 1, 27, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 55, 1, 17, 1, 0, 1, 1, 4, 0]

['no-recurrence-events', 45, 0, 22, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 45, 0, 12, 1, 0, 1, 1, 1, 0]

['no-recurrence-events', 45, 0, 37, 1, 0, 2, 1, 3, 0]

['no-recurrence-events', 55, 1, 22, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 35, 0, 17, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 45, 1, 22, 1, 0, 3, 0, 1, 0]

['no-recurrence-events', 35, 0, 12, 1, 0, 1, 1, 0, 0]

['no-recurrence-events', 65, 1, 17, 1, 0, 1, 0, 2, 0]

['no-recurrence-events', 65, 1, 22, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 55, 1, 17, 1, 0, 2, 1, 3, 0]

['no-recurrence-events', 55, 1, 42, 1, 0, 3, 0, 1, 0]

['no-recurrence-events', 55, 1, 32, 1, 0, 1, 1, 0, 0]

['no-recurrence-events', 65, 1, 12, 1, 0, 1, 1, 0, 0]

['no-recurrence-events', 75, 1, 12, 1, 0, 2, 0, 4, 0]

['no-recurrence-events', 35, 0, 32, 7, 1, 2, 1, 3, 0]

['no-recurrence-events', 35, 0, 27, 7, 1, 2, 1, 1, 1]

['no-recurrence-events', 55, 0, 27, 1, 1, 2, 0, 1, 0]

['no-recurrence-events', 45, 0, 37, 10, 1, 2, 1, 1, 1]

['no-recurrence-events', 45, 0, 37, 10, 1, 2, 1, 3, 1]

['no-recurrence-events', 45, 0, 42, 4, 1, 3, 1, 1, 1]

['no-recurrence-events', 45, 0, 32, 7, 0, 2, 0, 1, 0]

['no-recurrence-events', 55, 1, 42, 1, 0, 3, 0, 3, 0]

['no-recurrence-events', 65, 1, 32, 1, 0, 2, 0, 0, 1]

['no-recurrence-events', 35, 0, 22, 4, 0, 2, 1, 4, 0]

['no-recurrence-events', 35, 0, 42, 4, 0, 3, 1, 3, 1]

['no-recurrence-events', 45, 0, 7, 1, 0, 1, 0, 0, 1]

['no-recurrence-events', 35, 0, 42, 1, 0, 2, 0, 0, 1]

['no-recurrence-events', 45, 0, 32, 1, 0, 2, 0, 2, 0]

['no-recurrence-events', 55, 1, 42, 4, 1, 2, 0, 0, 0]

['no-recurrence-events', 55, 0, 22, 4, 1, 2, 0, 0, 0]

['no-recurrence-events', 65, 1, 12, 1, 0, 1, 0, 1, 0]

['no-recurrence-events', 45, 0, 47, 1, 0, 2, 0, 0, 1]

['no-recurrence-events', 65, 1, 47, 7, 1, 3, 0, 4, 0]

['no-recurrence-events', 45, 0, 27, 1, 1, 2, 0, 2, 1]

['no-recurrence-events', 65, 1, 52, 1, 0, 2, 1, 1, 1]

['no-recurrence-events', 55, 0, 32, 4, 1, 2, 0, 0, 1]

['no-recurrence-events', 35, 0, 22, 1, 0, 3, 0, 4, 0]

['no-recurrence-events', 55, 2, 32, 1, 0, 3, 1, 1, 0]

['no-recurrence-events', 55, 1, 27, 16, 1, 3, 1, 1, 0]

['no-recurrence-events', 65, 1, 32, 4, 1, 3, 0, 0, 0]

['no-recurrence-events', 55, 1, 37, 16, 0, 3, 0, 0, 0]

['no-recurrence-events', 65, 1, 17, 1, 0, 3, 1, 1, 1]

['no-recurrence-events', 35, 2, 17, 1, 0, 3, 1, 1, 0]

['no-recurrence-events', 65, 1, 42, 4, 0, 2, 1, 1, 1]

['no-recurrence-events', 55, 1, 27, 4, 1, 3, 1, 1, 0]

['no-recurrence-events', 55, 0, 32, 1, 0, 1, 0, 4, 0]

['no-recurrence-events', 55, 1, 32, 1, 0, 1, 1, 4, 0]

['no-recurrence-events', 45, 0, 37, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 45, 0, 27, 1, 0, 3, 1, 1, 1]

['no-recurrence-events', 45, 0, 32, 4, 1, 2, 1, 0, 0]

['no-recurrence-events', 65, 1, 12, 1, 0, 2, 1, 1, 1]

['no-recurrence-events', 65, 1, 27, 4, 1, 1, 1, 1, 1]

['no-recurrence-events', 65, 1, 27, 4, 1, 1, 1, 0, 1]

['no-recurrence-events', 45, 0, 22, 4, 0, 2, 1, 1, 0]

['no-recurrence-events', 45, 0, 22, 4, 0, 2, 1, 0, 0]

['no-recurrence-events', 45, 1, 42, 16, 1, 2, 1, 1, 1]

['no-recurrence-events', 55, 0, 12, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 45, 1, 32, 1, 0, 2, 0, 1, 1]

['no-recurrence-events', 35, 0, 22, 4, 1, 2, 1, 1, 1]

['no-recurrence-events', 35, 0, 17, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 65, 1, 32, 7, 1, 2, 1, 3, 0]

['no-recurrence-events', 55, 1, 22, 4, 1, 2, 1, 1, 0]

['no-recurrence-events', 55, 0, 27, 4, 1, 2, 0, 0, 1]

['no-recurrence-events', 45, 0, 32, 1, 0, 2, 1, 3, 1]

['no-recurrence-events', 45, 1, 27, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 65, 1, 12, 1, 0, 2, 0, 0, 0]

['no-recurrence-events', 55, 0, 27, 4, 0, 2, 1, 1, 1]

['no-recurrence-events', 45, 0, 22, 1, 0, 3, 1, 0, 1]

['no-recurrence-events', 45, 0, 37, 1, 1, 3, 1, 1, 1]

['no-recurrence-events', 45, 0, 37, 1, 1, 3, 1, 0, 1]

['no-recurrence-events', 45, 0, 27, 1, 0, 1, 1, 0, 1]

['no-recurrence-events', 55, 1, 32, 10, 1, 3, 0, 1, 1]

['no-recurrence-events', 55, 1, 32, 10, 1, 3, 0, 0, 1]

['no-recurrence-events', 45, 0, 22, 7, 0, 2, 1, 0, 1]

['no-recurrence-events', 55, 1, 27, 1, 0, 1, 0, 2, 0]

['no-recurrence-events', 65, 1, 17, 1, 0, 2, 0, 1, 1]

['no-recurrence-events', 45, 0, 12, 1, 0, 2, 1, 1, 0]

['no-recurrence-events', 55, 1, 22, 1, 1, 2, 1, 1, 0]

['no-recurrence-events', 45, 0, 17, 13, 0, 3, 1, 2, 1]

['no-recurrence-events', 45, 0, 27, 1, 0, 2, 0, 1, 1]

['no-recurrence-events', 55, 1, 32, 7, 1, 2, 0, 0, 0]

['no-recurrence-events', 35, 0, 12, 1, 0, 2, 0, 2, 0]

['no-recurrence-events', 55, 0, 52, 1, 1, 2, 1, 1, 1]

['no-recurrence-events', 55, 1, 37, 1, 0, 2, 0, 1, 0]

['no-recurrence-events', 55, 0, 12, 4, 0, 1, 1, 1, 0]

['no-recurrence-events', 45, 0, 12, 1, 0, 2, 0, 0, 1]

['no-recurrence-events', 55, 1, 17, 1, 1, 2, 0, 4, 1]

['no-recurrence-events', 55, 0, 27, 1, 0, 1, 0, 0, 0]

['no-recurrence-events', 65, 1, 27, 1, 0, 3, 1, 0, 0]

['recurrence-events', 55, 0, 17, 1, 0, 2, 0, 0, 0]

['recurrence-events', 45, 0, 42, 1, 0, 1, 0, 0, 0]

['recurrence-events', 55, 1, 37, 1, 0, 2, 0, 0, 0]

['recurrence-events', 55, 0, 27, 1, 0, 2, 0, 3, 0]

['recurrence-events', 35, 0, 2, 1, 0, 2, 1, 4, 0]

['recurrence-events', 55, 1, 32, 1, 0, 3, 0, 4, 0]

['recurrence-events', 55, 0, 27, 1, 0, 2, 0, 3, 0]

['recurrence-events', 55, 0, 32, 1, 0, 3, 0, 3, 0]

['recurrence-events', 45, 0, 37, 1, 0, 1, 1, 1, 0]

['recurrence-events', 45, 0, 22, 1, 0, 2, 0, 0, 0]

['recurrence-events', 55, 1, 22, 1, 0, 2, 1, 4, 0]

['recurrence-events', 45, 0, 32, 1, 0, 3, 1, 3, 0]

['recurrence-events', 55, 0, 27, 1, 0, 1, 1, 1, 0]

['recurrence-events', 65, 1, 42, 1, 0, 2, 1, 0, 0]

['recurrence-events', 45, 1, 22, 1, 0, 2, 1, 1, 0]

['recurrence-events', 55, 1, 22, 1, 0, 2, 0, 1, 0]

['recurrence-events', 45, 0, 17, 1, 0, 2, 0, 1, 0]

['recurrence-events', 65, 1, 32, 1, 0, 3, 1, 4, 0]

['recurrence-events', 35, 0, 17, 1, 0, 1, 1, 0, 0]

['recurrence-events', 45, 0, 27, 1, 0, 3, 0, 3, 0]

['recurrence-events', 35, 0, 32, 1, 0, 1, 1, 1, 0]

['recurrence-events', 65, 1, 27, 1, 0, 3, 0, 2, 1]

['recurrence-events', 65, 1, 22, 1, 0, 3, 1, 0, 0]

['recurrence-events', 35, 0, 27, 4, 1, 3, 0, 0, 1]

['recurrence-events', 45, 1, 22, 4, 0, 3, 1, 0, 1]

['recurrence-events', 45, 0, 32, 16, 1, 3, 0, 0, 0]

['recurrence-events', 55, 0, 32, 1, 0, 3, 1, 1, 1]

['recurrence-events', 65, 1, 42, 4, 1, 3, 1, 0, 0]

['recurrence-events', 65, 1, 47, 1, 0, 1, 1, 3, 1]

['recurrence-events', 55, 0, 52, 10, 1, 2, 1, 1, 0]

['recurrence-events', 45, 0, 32, 4, 0, 2, 1, 1, 0]

['recurrence-events', 35, 0, 32, 4, 0, 3, 1, 1, 1]

['recurrence-events', 75, 1, 17, 10, 1, 1, 0, 0, 1]

['recurrence-events', 65, 1, 32, 1, 0, 3, 1, 1, 1]

['recurrence-events', 55, 0, 27, 4, 1, 3, 0, 0, 1]

['recurrence-events', 45, 0, 27, 1, 0, 2, 1, 0, 0]

['recurrence-events', 45, 0, 27, 1, 0, 2, 1, 0, 0]

['recurrence-events', 35, 0, 37, 1, 0, 3, 0, 0, 0]

['recurrence-events', 45, 0, 22, 4, 1, 2, 1, 3, 1]

['recurrence-events', 65, 1, 22, 4, 0, 2, 0, 0, 1]

['recurrence-events', 45, 0, 17, 16, 1, 3, 0, 0, 0]

['recurrence-events', 55, 1, 27, 7, 0, 3, 0, 0, 1]

['recurrence-events', 55, 1, 22, 4, 1, 3, 1, 3, 0]

['recurrence-events', 45, 0, 32, 13, 1, 3, 0, 1, 1]

['recurrence-events', 35, 0, 32, 10, 0, 2, 1, 1, 1]

['recurrence-events', 35, 0, 17, 7, 1, 3, 0, 0, 1]

['recurrence-events', 55, 1, 32, 10, 1, 3, 0, 2, 1]

['recurrence-events', 65, 1, 37, 7, 1, 3, 0, 0, 0]

['recurrence-events', 35, 0, 22, 4, 1, 2, 0, 0, 0]

['recurrence-events', 45, 0, 27, 1, 0, 3, 0, 1, 0]

['recurrence-events', 45, 0, 52, 1, 0, 2, 1, 0, 1]

['recurrence-events', 35, 0, 42, 1, 0, 1, 0, 1, 0]

['recurrence-events', 65, 1, 52, 1, 0, 3, 1, 1, 0]

['recurrence-events', 45, 0, 32, 1, 1, 3, 1, 3, 0]

['recurrence-events', 45, 0, 32, 7, 1, 3, 1, 1, 0]

['recurrence-events', 45, 0, 32, 1, 0, 1, 0, 0, 1]

['recurrence-events', 45, 0, 22, 4, 1, 2, 0, 0, 1]

['recurrence-events', 55, 1, 32, 7, 1, 2, 0, 2, 1]

['recurrence-events', 55, 1, 32, 4, 0, 3, 1, 1, 0]

['recurrence-events', 65, 1, 27, 4, 0, 2, 1, 3, 0]

['recurrence-events', 45, 1, 27, 13, 1, 3, 0, 2, 1]

['recurrence-events', 65, 1, 27, 1, 0, 3, 0, 1, 0]

['recurrence-events', 55, 2, 22, 1, 1, 1, 0, 1, 0]

['recurrence-events', 55, 2, 22, 1, 1, 1, 0, 0, 0]

['recurrence-events', 35, 0, 37, 10, 1, 3, 0, 0, 0]

['recurrence-events', 45, 0, 32, 4, 1, 2, 0, 3, 0]

['recurrence-events', 65, 1, 22, 25, 1, 3, 0, 0, 1]

['recurrence-events', 35, 0, 37, 1, 0, 3, 0, 0, 0]

['recurrence-events', 45, 0, 27, 1, 0, 2, 0, 0, 1]

['recurrence-events', 35, 0, 32, 1, 0, 2, 0, 1, 0]

['recurrence-events', 35, 0, 22, 1, 0, 3, 0, 1, 1]

['recurrence-events', 65, 1, 22, 1, 0, 1, 1, 1, 0]

['recurrence-events', 45, 1, 32, 4, 0, 3, 0, 0, 0]

['recurrence-events', 55, 1, 32, 4, 0, 3, 0, 0, 0]

Decision-Tree Classifier

Accuracy is 61.111 with 75% of training data

Accuracy is 64.583 with 66.6% of training data

Accuracy: [60.3448275862069, 64.08045977011494, 62.84722222222222]

Naive-Bayes Classifier

Accuracy is 70.83333333333334 with 75% of Training Data

Accuracy is 70.83333333333334 with 66.6% of Training Data

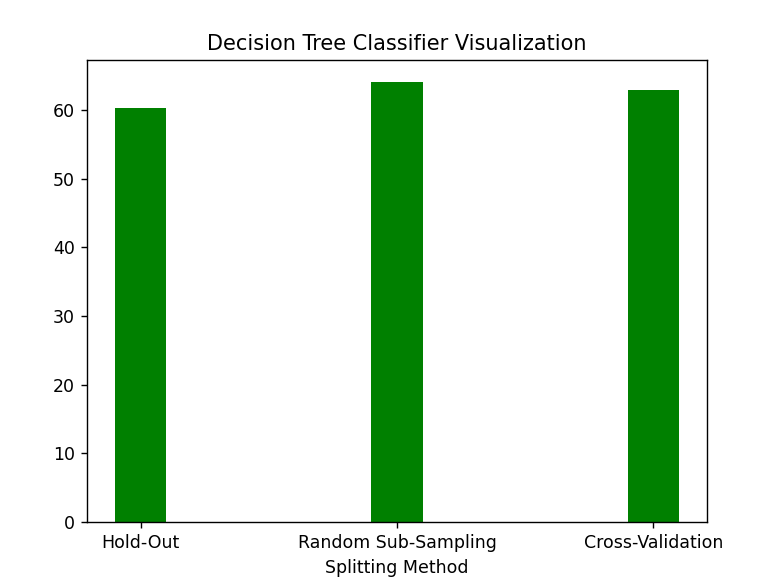
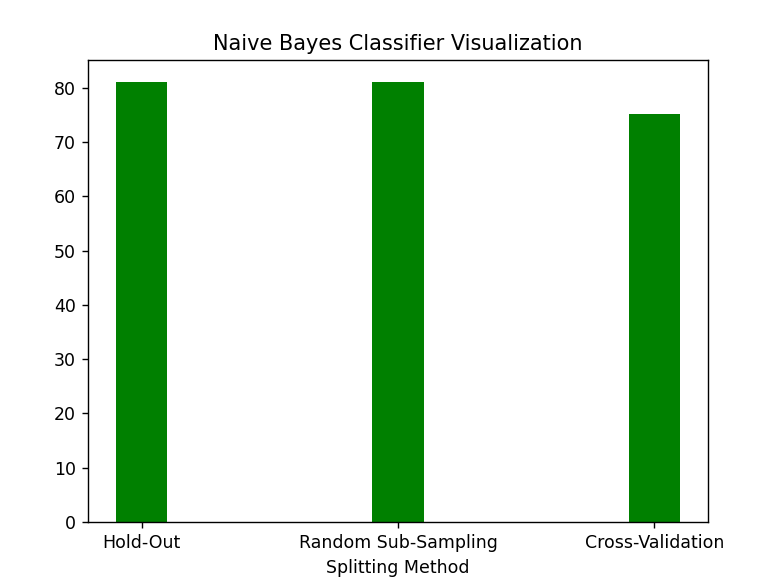
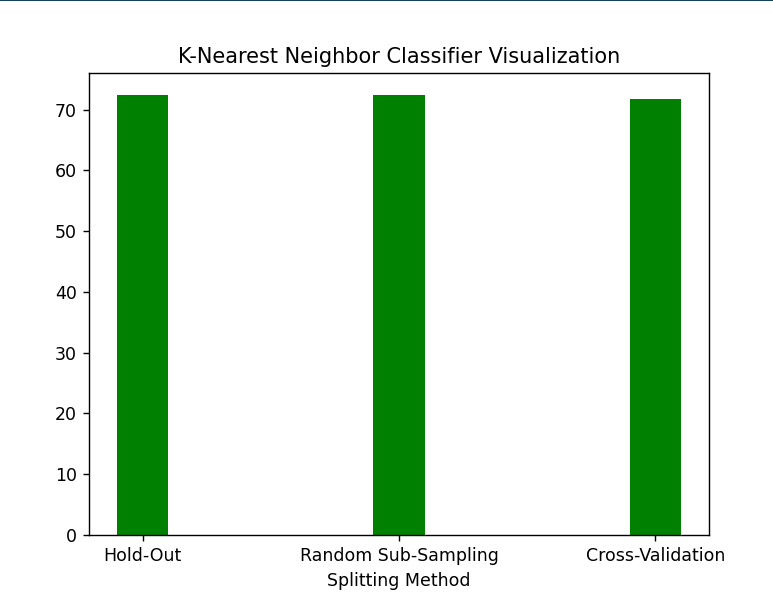
Accuracy: [81.03448275862068, 81.03448275862068, 75.13440860215054]

K-Nearest Neighbour

Accuracy is 73.61111111111111 with 75% of Training Data

Accuracy is 71.875 with 66.6% of Training Data

Accuracy: [72.41379310344827, 72.41379310344827, 71.68458781362007]

Q6. Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms.

Source Code:

k-Means

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data = pd.read\_csv('HTRU\_2.csv')[:1000]

wcss=[]

for i in range(1,20):

    km=KMeans(n\_clusters=i)

    km.fit\_predict(data)

    wcss.append(km.inertia\_)

plt.plot(range(1,20),wcss)

plt.show()

X=data.iloc[:,:].values

km = KMeans(n\_clusters=6)

Y\_means = km.fit\_predict(X)

first=0

second=5

print(X[Y\_means==0,first])

plt.scatter(X[Y\_means==0,first],X[Y\_means==0,second],color="red")

plt.scatter(X[Y\_means==1,first],X[Y\_means==1,second],color="blue")

plt.scatter(X[Y\_means==2,first],X[Y\_means==2,second],color="yellow")

plt.scatter(X[Y\_means==3,first],X[Y\_means==3,second],color="green")

plt.scatter(X[Y\_means==4,first],X[Y\_means==4,second],color="brown")

plt.show()

DBScan

from sklearn.cluster import DBSCAN

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

data = pd.read\_csv('HTRU\_2.csv')[:1000]

X=np.array([[i,j] for i,j in zip(data.values[:,0],data.values[:,5])])

clustering = DBSCAN(eps=6,min\_samples=5).fit(X)

print(clustering.labels\_)

plt.scatter(X[clustering.labels\_==0,0],X[clustering.labels\_==0,1],color="red")

plt.scatter(X[clustering.labels\_==1,0],X[clustering.labels\_==1,1],color="blue")

plt.scatter(X[clustering.labels\_==2,0],X[clustering.labels\_==2,1],color="yellow")

plt.scatter(X[clustering.labels\_==3,0],X[clustering.labels\_==3,1],color="green")

plt.scatter(X[clustering.labels\_==4,0],X[clustering.labels\_==4,1],color="brown")

plt.show()

Hierarchical

import numpy as np

import pandas as pd

import scipy.cluster.hierarchy as shc

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

data = pd.read\_csv('HTRU\_2.csv')[:1000]

plt.figure(figsize=(10,7))

plt.title("Dendogram")

X=np.array([[i,j] for i,j in zip(data.values[:,0],data.values[:,5])])

dend = shc.dendrogram(shc.linkage(X[:,0:2],method="ward"))

plt.show()

cluster = AgglomerativeClustering(n\_clusters=4,affinity="euclidean",linkage="ward")

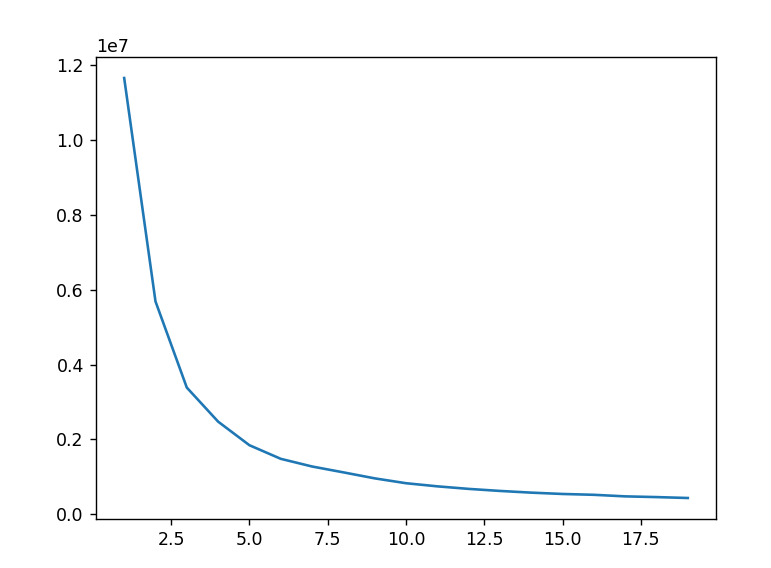
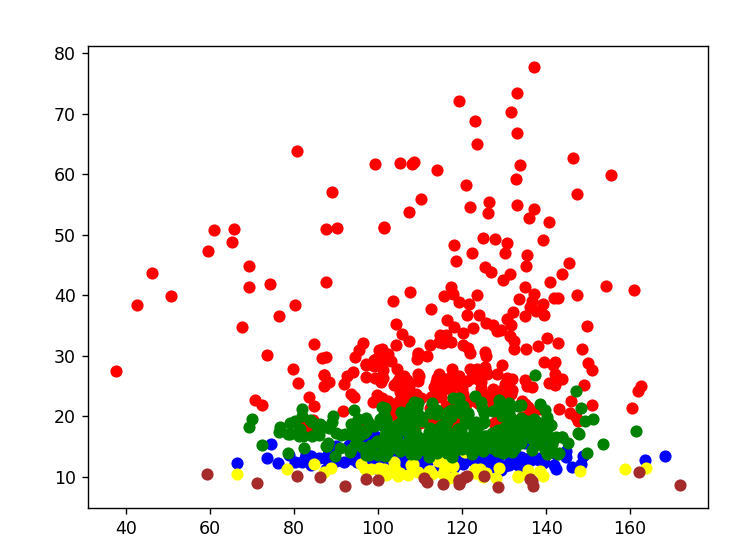
labels\_=cluster.fit\_predict(X[:,0:])

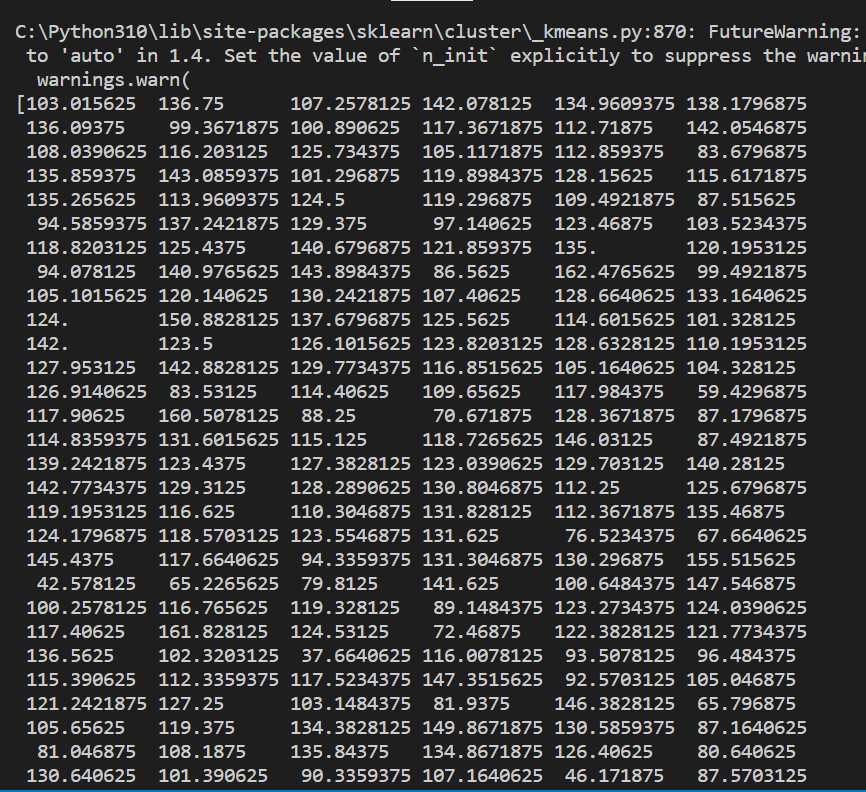
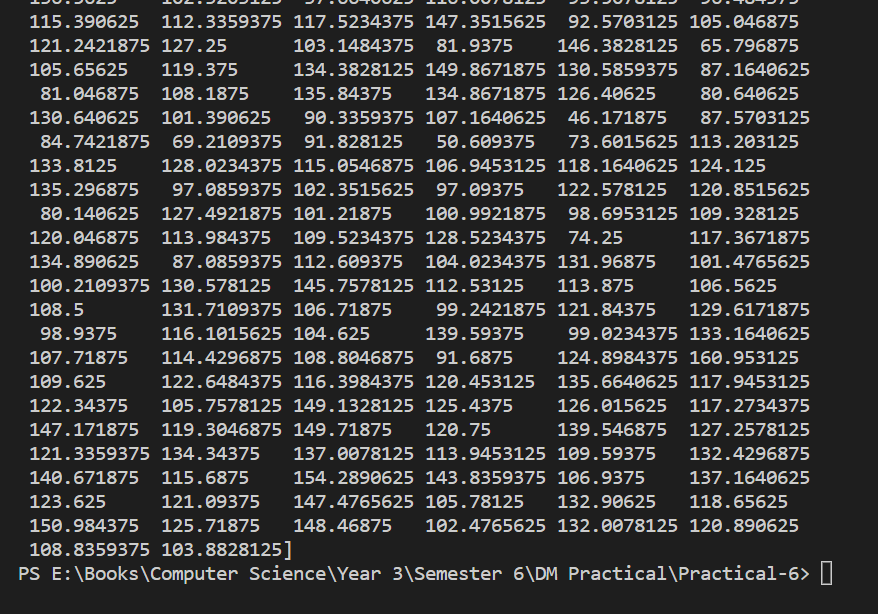
plt.scatter(X[:,0],X[:,1],c=cluster.labels\_,cmap="rainbow")

plt.show()

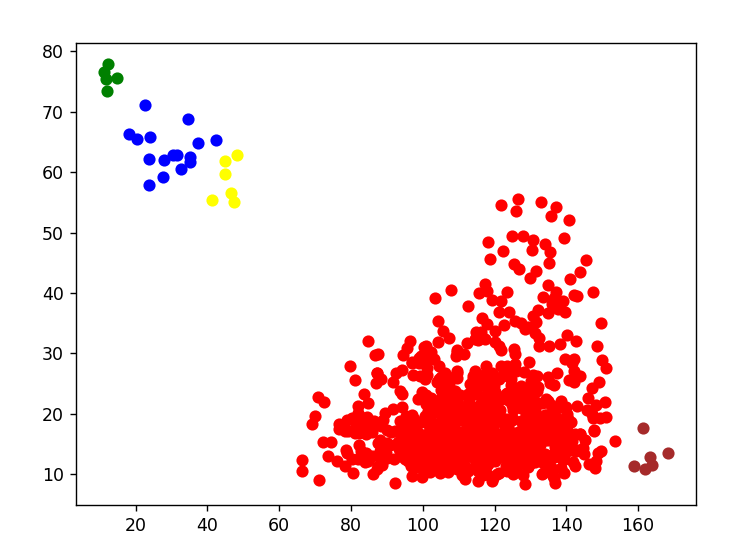
Output:

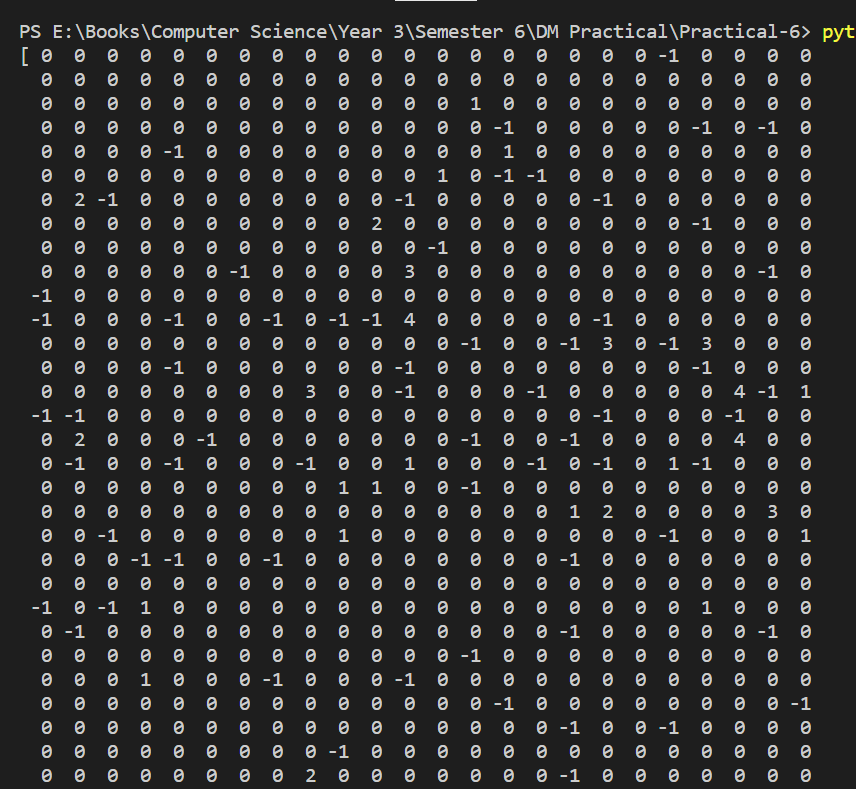
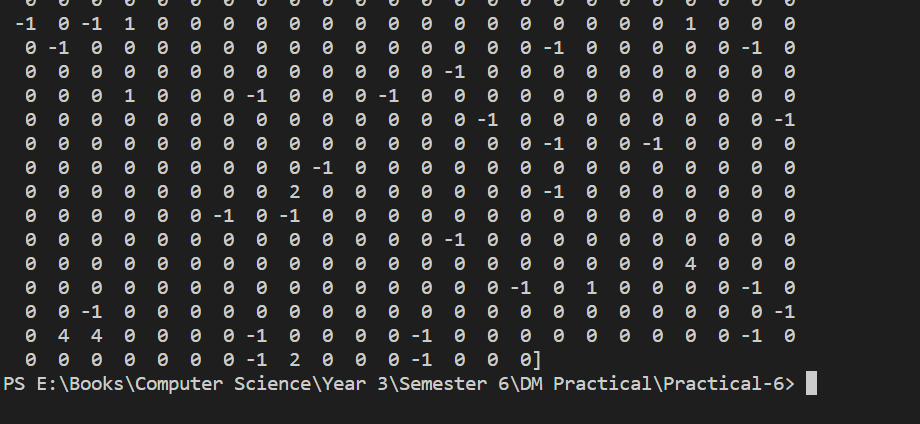
k-means

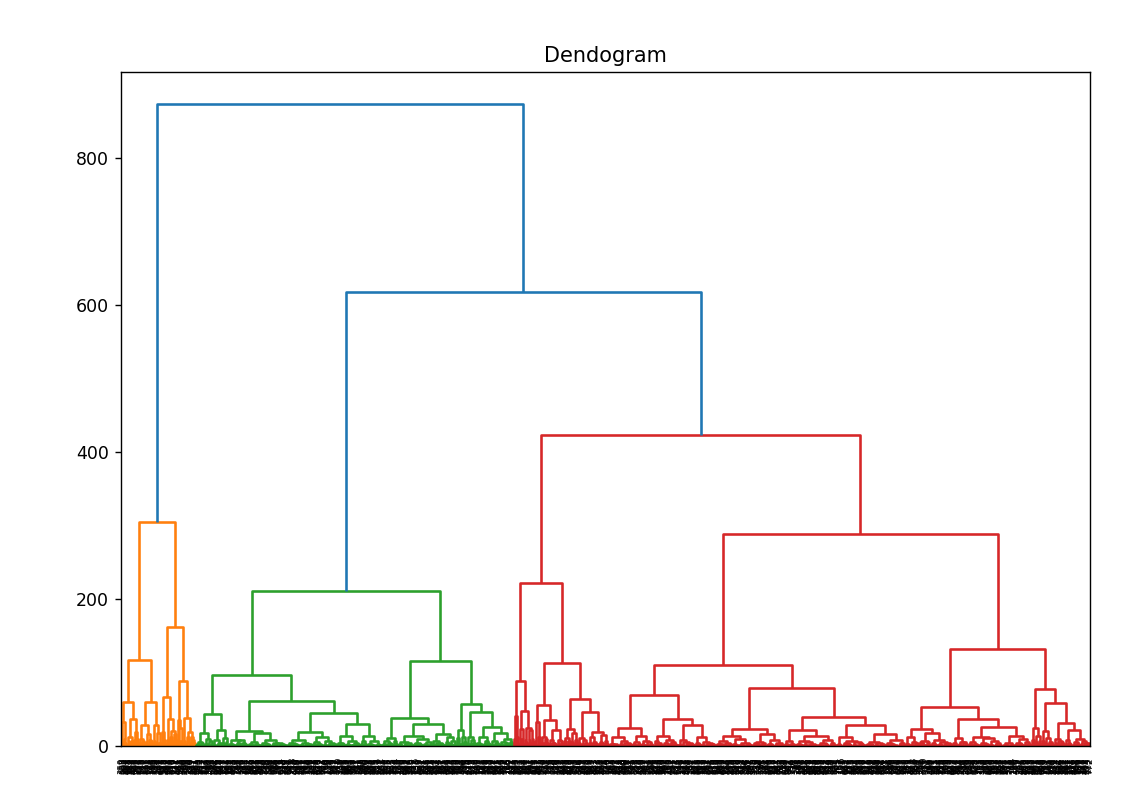
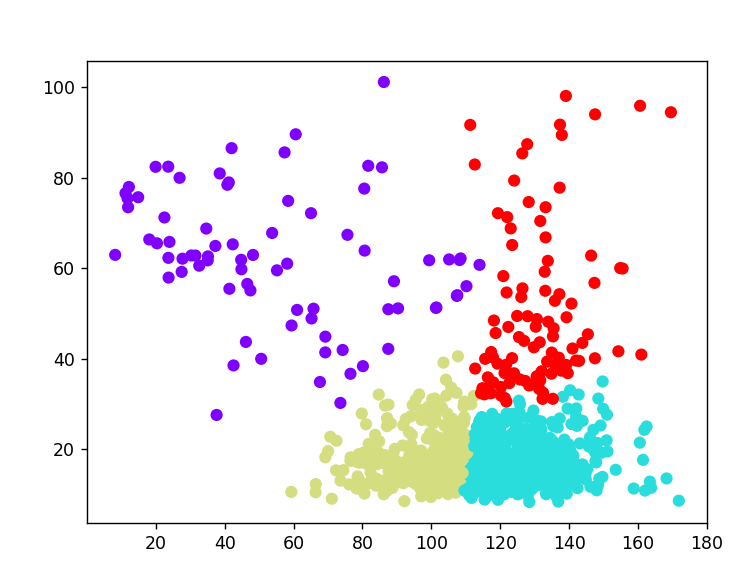
 

DBScan



Heirarchical

Q7. Students should be promoted to take up one project on any UCI/kaggle/data.gov.in or a dataset verified by the teacher. Preprocessing steps and at least one data mining technique should be shown on the selected dataset. This will allow the students to have a practical knowledge of how to apply the various skills learnt in the subject for a single problem/project.

Source Code:

# Students should be promoted to take up one project on any UCI/kaggle/data.gov.in or a dataset

# verified by the teacher. Preprocessing steps and at least one data mining technique should be shown

# on the selected dataset. This will allow the students to have a practical knowledge of how to apply

# the various skills learnt in the subject for a single problem/project.

import pandas as pd

from sklearn import preprocessing

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

print("Data Set : Dry\_Bean.")

data = pd.read\_csv('Dry\_Bean.csv')

X = data.values[:, 0:16]

Y = data.values[:, 16]

# Applying preprocessing technique that is standardization

scaler = preprocessing.StandardScaler().fit(X)

# Applying Scaler tranformation

X = scaler.transform(X)

# Splitting the data into training and testing data using hold out method

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

    X, Y, test\_size=0.25, shuffle=True)

decision\_Tree = DecisionTreeClassifier()

# Training the model on training data set

decision\_Tree.fit(X\_train, Y\_train)

# Applying the model on the testing data set

Y\_predicted = decision\_Tree.predict(X\_test)

# Computing the accuracy of the decision tree classifier model

print(("Accuracy is "), accuracy\_score(Y\_test, Y\_predicted) \* 100,

      ("when using Decision Tree with 75 % of training data"))

Output:

