DATA MINING PRACTICAL FILE

Abhineet Raman(2002078)

B.SC.(H.) COMPUTER SCIENCE Semester: 6

Index

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:

```
2] No such file or directory
PS E:\Books\Computer Science\Year 3\Semester 6\DM P
People-1:-
Age:  satisfied
AgeGroup:  satisfied
Status:  satisfied
Year-Married:  satisfied
[1, 0]
[1, 0]
[1, 0]
[1, 0]
People-2:-
Age:  satisfied
AgeGroup:  satisfied
Status:  satisfied
Year-Married:  satisfied
[2, 0]
[2, 0]
[2, 0]
[2, 0]
People-3:-
Age:  satisfied
AgeGroup:  satisfied
Status:  satisfied
Year-Married:  unsatisfied
[3, 0]
```

```
People-3:-
Age:  satisfied
AgeGroup:  satisfied
Status:  satisfied
Year-Married:  unsatisfied
[3, 0]
[3, 0]
[3, 0]
[2, 1]
People-4:-
Age:  unsatisfied
AgeGroup:  satisfied
Status:  satisfied
Year-Married:  satisfied
[3, 1]
[5, 0]
[4, 1]
People-1: All Satisfied
People-2: All Satisfied
People-3: One or more condition violates
People-4: One or more condition violates
People-5: One or more condition violates
PS E:\Books\Computer Science\Year 3\Semester
```

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:

```
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical> python Practical_2.py
Complete Observations:  96
InComplete Observations:  54
Complete Observation Percentage:  64.0 %
InComplete Observation Percentage:  36.0 %
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
Iris with Sepal Length more than 30cm:  13
Iris with its Petal Length equal to atleast two times of its Petal Width:  129
Iris with its Petal Length equal to less than two times of its Petal Width:  21
Iris with Sepal greater than its Petals:  138
Iris with Sepal less than its Petals:  12
```

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:

```
PS E:\Books\Computer Science
False
False
[6.6, 3.4, 4.4, 1.2]
[6.8, 3.3, 5.8, 0.8]
```

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:

```
FileNotFoundError: [Errno 2] No such file or directory: './breast-cancer.csv'
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical\Practical 4> python Prac-4.py
       A      B      C      D      E      F      H
0   True   True   True   True  False   True   True
1  False   True  False  False   True   True   True
2   True  False   True  False   True  False  False
3  False   True   True   True  False   True   True
4   True   True   True   True   True  False  False
7     (C, D)         (H)               0.625               0.750    0.500    0.800000  1.066667   0.03125        1.25
8        (D)      (H, C)               0.625               0.500    0.500    0.800000  1.600000   0.18750        2.50
   support itemsets
0    0.750      (C)
1    0.625      (D)
2    0.750      (H)
3    0.625   (C, D)
  antecedents consequents  antecedent support  consequent support  support  confidence      lift  leverage  conviction
0         (C)         (D)               0.750               0.625    0.625    0.833333  1.333333   0.15625        2.25
1         (D)         (C)               0.625               0.750    0.625    1.000000  1.333333   0.15625         inf
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical\Practical 4>
```

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
```

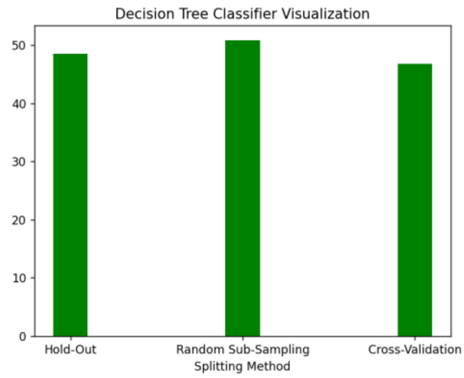
```
plt.bar(Xval,Yval,color="green",width=0.2)

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
```

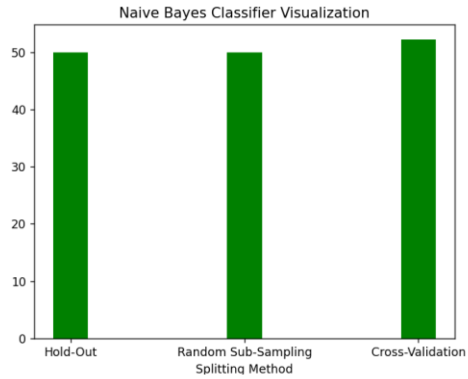
```
for i in range(0,k):
                                X_train,X_test,Y_train,Y_test                =
train_test_split(X,Y,test_size=0.2,random_state=1000,shuffle=True,stratify=Y)
   X_train = scale.fit_transform(X_train)
   X_test = scale.fit_transform(X_test)
   Accuracy_Random.append(get_Score(knn,X_train,Y_train,X_test,Y_test))
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()
```

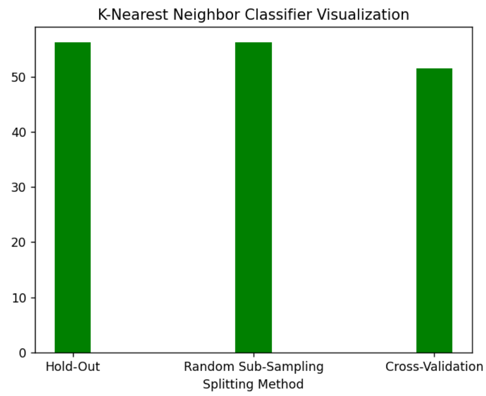
Output:

1.

```
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical\Practical-5> p
Decision Tree Classifier
Accuracy is 51.86602870813397 with 75% of training data
Accuracy is 48.09489575844716 with 66.6% of training data
Accuracy:  [48.44497607655502, 50.79744816586922, 46.75683887446957]
Naive-Bayes Classifier
Accuracy is 51.770334928229666 with 75% of Training Data
Accuracy is 52.480230050323506 with 66.6% of Training Data
Accuracy:  [50.0, 50.0, 52.215321550694185]
K-Nearest Neighbour
Accuracy is 53.110047846889955 with 75% of Training Data
Accuracy is 55.068296189791525 with 66.6% of Training Data
Accuracy:  [56.2200956937799, 56.2200956937799, 51.5460594075722]
```







2.

```
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical\Pra
['no-recurrence-events', 35, 0, 32, 1, 0, 3, 0, 0, 0]
['no-recurrence-events', 45, 0, 22, 1, 0, 2, 1, 3, 0]
['no-recurrence-events', 45, 0, 22, 1, 0, 2, 0, 0, 0]
['no-recurrence-events', 65, 1, 17, 1, 0, 2, 1, 1, 0]
['no-recurrence-events', 45, 0, 2, 1, 0, 2, 1, 2, 0]
['no-recurrence-events', 65, 1, 17, 1, 0, 2, 0, 0, 0]
['no-recurrence-events', 55, 0, 27, 1, 0, 2, 0, 0, 0]
['no-recurrence-events', 65, 1, 22, 1, 0, 1, 0, 0, 0]
['no-recurrence-events', 45, 0, 52, 1, 0, 2, 0, 0, 0]
['no-recurrence-events', 45, 0, 22, 1, 0, 2, 1, 1, 0]
['no-recurrence-events', 45, 0, 2, 1, 0, 3, 0, 4, 0]
['no-recurrence-events', 55, 1, 27, 1, 0, 2, 0, 0, 0]
['no-recurrence-events', 65, 2, 12, 1, 0, 1, 0, 3, 0]
['no-recurrence-events', 55, 1, 27, 1, 0, 3, 0, 3, 0]
['no-recurrence-events', 45, 0, 32, 1, 0, 3, 0, 1, 0]
['no-recurrence-events', 65, 2, 32, 1, 0, 1, 0, 0, 0]
['no-recurrence-events', 45, 0, 17, 1, 0, 2, 0, 0, 0]
['no-recurrence-events', 55, 0, 32, 1, 0, 3, 0, 0, 0]
['no-recurrence-events', 65, 1, 32, 1, 0, 3, 0, 0, 0]
['no-recurrence-events', 55, 1, 32, 1, 0, 1, 1, 3, 0]
['no-recurrence-events', 55, 1, 42, 1, 0, 2, 0, 0, 0]
['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', 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

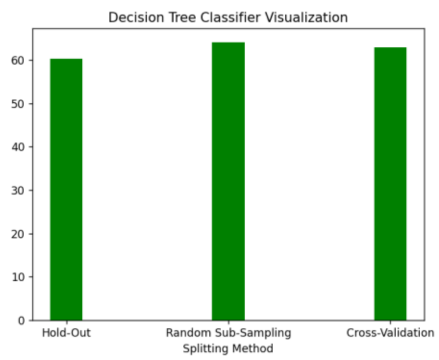
Accuracy: [81.03448275862068, 81.03448275862068, 75.13440860215054]

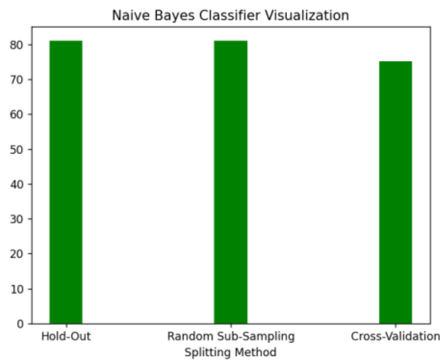
K-Nearest Neighbour

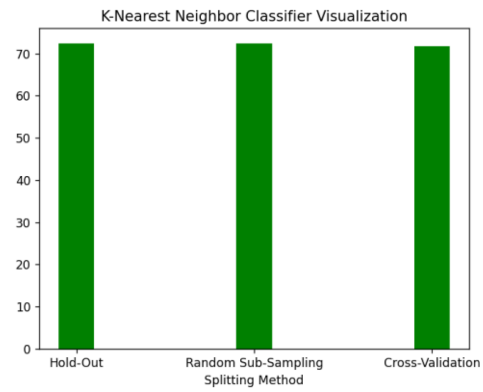
Accuracy is 73.61111111111111 with 75% of Training Data

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()
```

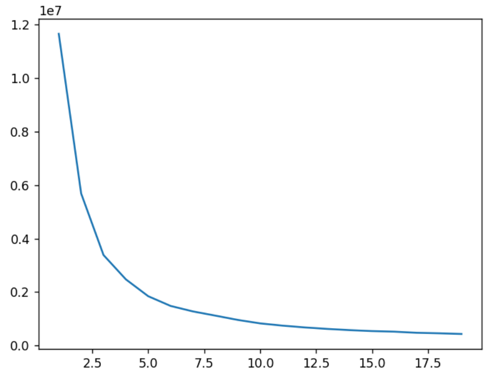
Hierarchical

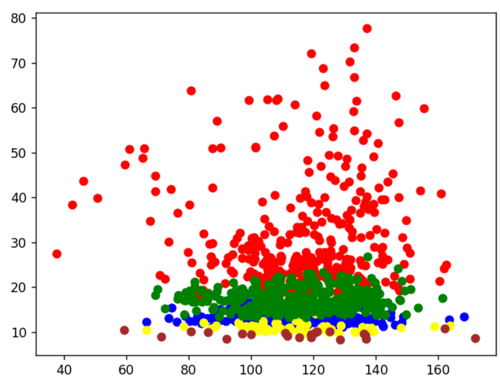
```
import numpy as np
import pandas as pd
import scipy.cluster.hierarchy as shc
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
```

```
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

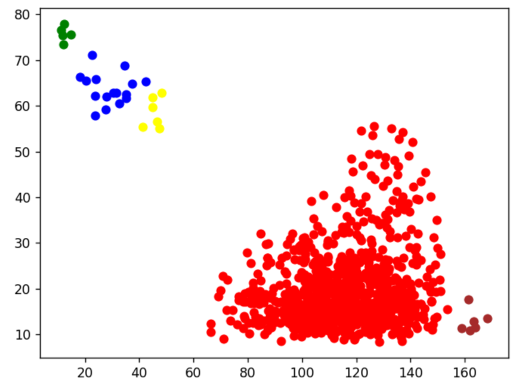




```
C:\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:
 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warni
  warnings.warn(
[103.015625  136.75      107.2578125 142.078125  134.9609375 138.1796875
 136.09375    99.3671875 100.890625  117.3671875 112.71875   142.0546875
 108.0390625 116.203125  125.734375  105.1171875 112.859375   83.6796875
 135.859375  143.0859375 101.296875  119.8984375 128.15625   115.6171875
 135.265625  113.9609375 124.5       119.296875  109.4921875  87.515625
  94.5859375 137.2421875 129.375      97.140625  123.46875   103.5234375
 118.8203125 125.4375    140.6796875 121.859375  135.        120.1953125
  94.078125  140.9765625 143.8984375  86.5625    162.4765625  99.4921875
 105.1015625 120.140625  130.2421875 107.40625   128.6640625 133.1640625
 124.        150.8828125 137.6796875 125.5625    114.6015625 101.328125
 142.        123.5       126.1015625 123.8203125 128.6328125 110.1953125
 127.953125  142.8828125 129.7734375 116.8515625 105.1640625 104.328125
 126.9140625  83.53125   114.40625   109.65625   117.984375   59.4296875
 117.90625   160.5078125  88.25       70.671875  128.3671875  87.1796875
 114.8359375 131.6015625 115.125     118.7265625 146.03125    87.4921875
 139.2421875 123.4375    127.3828125 123.0390625 129.703125  140.28125
 142.7734375 129.3125    128.2890625 130.8046875 112.25      125.6796875
 119.1953125 116.625     110.3046875 131.828125  112.3671875 135.46875
 124.1796875 118.5703125 123.5546875 131.625      76.5234375  67.6640625
 145.4375    117.6640625  94.3359375 131.3046875 130.296875  155.515625
  42.578125   65.2265625  79.8125    141.625     100.6484375 147.546875
 100.2578125 116.765625  119.328125   89.1484375 123.2734375 124.0390625
 117.40625   161.828125  124.53125    72.46875   122.3828125 121.7734375
 136.5625    102.3203125  37.6640625 116.0078125  93.5078125  96.484375
 115.390625  112.3359375 117.5234375 147.3515625  92.5703125 105.046875
 121.2421875 127.25      103.1484375  81.9375    146.3828125  65.796875
 105.65625   119.375     134.3828125 149.8671875 130.5859375  87.1640625
  81.046875  108.1875    135.84375   134.8671875 126.40625    80.640625
 130.640625  101.390625   90.3359375 107.1640625  46.171875   87.5703125
```

```
 115.390625  112.3359375 117.5234375 147.3515625  92.5703125 105.046875
 121.2421875 127.25      103.1484375  81.9375    146.3828125  65.796875
 105.65625   119.375     134.3828125 149.8671875 130.5859375  87.1640625
  81.046875  108.1875    135.84375   134.8671875 126.40625    80.640625
 130.640625  101.390625   90.3359375 107.1640625  46.171875   87.5703125
  84.7421875  69.2109375  91.828125   50.609375   73.6015625 113.203125
 133.8125    128.0234375 115.0546875 106.9453125 118.1640625 124.125
 135.296875   97.0859375 102.3515625  97.09375   122.578125  120.8515625
  80.140625  127.4921875 101.21875   100.9921875  98.6953125 109.328125
 120.046875  113.984375  109.5234375 128.5234375  74.25      117.3671875
 134.890625   87.0859375 112.609375  104.0234375 131.96875   101.4765625
 100.2109375 130.578125  145.7578125 112.53125   113.875     106.5625
 108.5       131.7109375 106.71875    99.2421875 121.84375   129.6171875
  98.9375    116.1015625 104.625     139.59375    99.0234375 133.1640625
 107.71875   114.4296875 108.8046875  91.6875    124.8984375 160.953125
 109.625     122.6484375 116.3984375 120.453125  135.6640625 117.9453125
 122.34375   105.7578125 149.1328125 125.4375    126.015625  117.2734375
 147.171875  119.3046875 149.71875   120.75      139.546875  127.2578125
 121.3359375 134.34375   137.0078125 113.9453125 109.59375   132.4296875
 140.671875  115.6875    154.2890625 143.8359375 106.9375    137.1640625
 123.625     121.09375   147.4765625 105.78125   132.90625   118.65625
 150.984375  125.71875   148.46875   102.4765625 132.0078125 120.890625
 108.8359375 103.8828125]
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical\Practical-6>
```

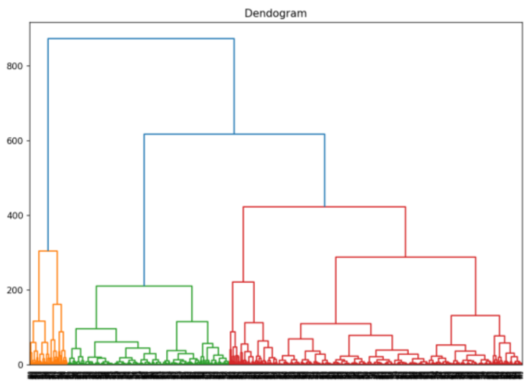
DBScan

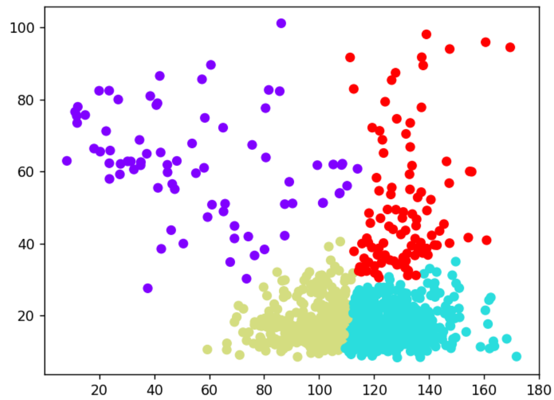



```
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical\Practical-6> pyt
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0 -1  0 -1  0
  0  0  0  0 -1  0  0  0  0  0  0  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  1  0 -1 -1  0  0  0  0  0  0  0  0
  0  2 -1  0  0  0  0  0  0  0  0 -1  0  0  0  0  0 -1  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0 -1  0  0  0
  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  0  0  0  0  0 -1  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0 -1  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
 -1  0  0  0 -1  0  0 -1  0 -1 -1  4  0  0  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 -1  3  0 -1  3  0  0  0
  0  0  0  0 -1  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0 -1  0  0  0
  0  0  0  0  0  0  0  0  3  0  0 -1  0  0  0 -1  0  0  0  0  0  4 -1  1
 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0 -1  0  0
  0  2  0  0  0 -1  0  0  0  0  0  0  0 -1  0  0 -1  0  0  0  0  4  0  0
  0 -1  0  0 -1  0  0  0 -1  0  0  1  0  0  0 -1  0 -1  0  1 -1  0  0  0
  0  0  0  0  0  0  0  0  0  1  1  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  0  1  2  0  0  0  0  3  0
  0  0 -1  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0 -1  0  0  0  1
  0  0  0 -1 -1  0  0 -1  0  0  0  0  0  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  0  0  0  0  0  0
 -1  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0 -1  0
  0  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  0  1  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  0 -1  0  0  0  0  0  0  0  0 -1
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0 -1  0  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  0  0
  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0
```

```
 -1  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0 -1  0
  0  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  0  1  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  0 -1  0  0  0  0  0  0  0  0 -1
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0 -1  0  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  0  0
  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0
  0  0  0  0  0  0 -1  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  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  0  0  0  0  0  4  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  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  0 -1
  0  4  4  0  0  0  0 -1  0  0  0  0 -1  0  0  0  0  0  0  0  0  0 -1  0
  0  0  0  0  0  0  0 -1  2  0  0  0 -1  0  0  0]
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical\Practical-6>
```

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:

```
PS E:\Books\Computer Science\Year 3\Semester 6\DM Practical\Practical-7> python Practi
Data Set : Dry_Bean.
Accuracy is  89.89127240669997 when using Decision Tree with 75 % of training data
```