**Section 1: Preprocessing.**

**Ques 1: Create a file “people.txt” with the following data:**

1. **Read the data from the file “people.txt”.**
2. **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)**

**CODE:**

import pandas as pd

from matplotlib import pyplot as plt

df = {

    "age": [21,2,18,224,34],

"agegroup": ["adult","child","adult","elderly","adult"],

"height": [6.0,3,5.7,5,-7],

"status": ["Single","Married","Married","Widowed","Married"],

"yearsmarried": [-1,0,20,2,3]}

df

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

df = pd.DataFrame(df)

df

df.to\_csv('people.csv')

df = pd.read\_csv('people.csv', index\_col=0)

df

#(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.

def d1():

    print("Age range:")

    print((df['age']<150) & (df['age']>-1))

def d2():

    print("Years Married:")

    print(df['age'] > df['yearsmarried'])

def d3():

    print("Status:")

    print((df["status"]=="Married") | (df["status"]=="Single") | (df["status"]=="Widowed"))

def d4():

    df.loc[df['age']<18, 'agegroup'] = 'child'

    df.loc[(df['age']>=18) & (df['age']<65), 'agegroup'] = 'adult'

    df.loc[df['age']>=65, 'agegroup'] = 'elderly'

    print("Age group:")

    print(df)

#(iv)Summarize the results obtained in part (iii)

x=(df["status"]=="Married") | (df["status"]=="Single") | (df["status"]=="Widowed")

print("no.of valid values:",x.sum())

print("no.of valid values:",len(x)-x.sum())

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

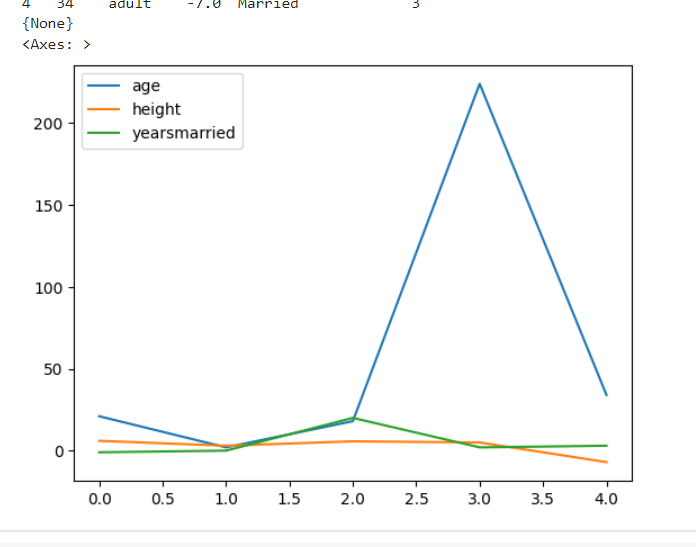
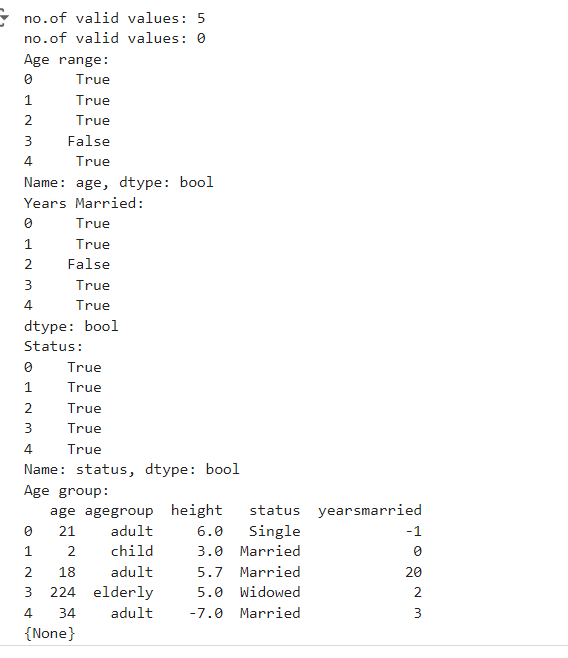
E = {d1(),d2(),d3(),d4()}

print(E)

#(v)Visualize the results obtained in part (iii)

df.plot()

**OUTPUT:**

****

**Ques 2: Perform the following preprocessing tasks on the dirty\_iris datasetii.**

1. **Calculate the number and percentage of observations that are complete.**
2. **Replace all the special values in data with NA.**
3. **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.**

1. **Determine how often each rule is broken (violatedEdits). Also summarize and plot the result**
2. **. v) Find outliers in sepal length using boxplot and boxplot.stats**

**CODE:**

**OUTPUT:**

**Ques 3: 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.**

**CODE(wine dataset):**

from sklearn.datasets import load\_wine

from sklearn import preprocessing

import pandas as pd

#load the wine dataset

wine = load\_wine()

print(wine)

#seperate the data and target attributes

X = wine.data

y = wine.target

print("MEAN OF THE WINE DATASET:", int(X.mean()))

print("STANDARD DEVIATION OF THE WINE DATASET:",

      X.std())

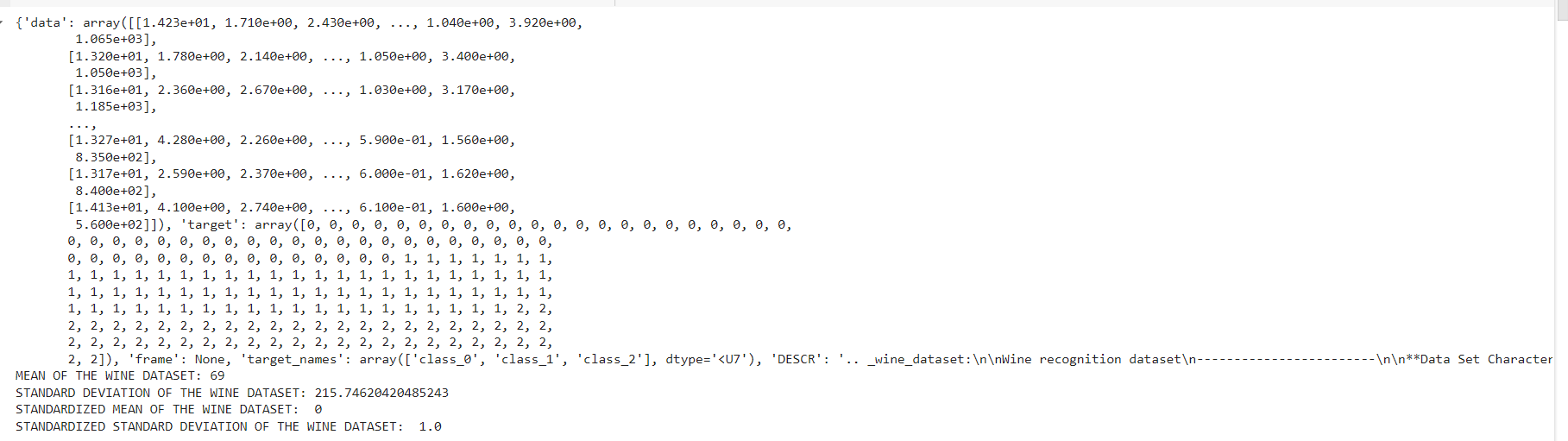
#standarize the data attributes

standardized\_X = preprocessing.scale(X)

print("STANDARDIZED MEAN OF THE WINE DATASET: ", int(standardized\_X.mean()))

print("STANDARDIZED STANDARD DEVIATION OF THE WINE DATASET: ", standardized\_X.std())

**OUTPUT:**

****

**CODE(iris dataset):**

from sklearn.datasets import load\_iris

from sklearn import preprocessing

import pandas as pd

#load the iris dataset

iris = load\_iris()

print(iris)

#seperate the data and target attributes

X = iris.data

y =  iris.target

print("MEAN OF THE IRIS DATASET:", int(X.mean()))

print("STANDARD DEVIATION OF THE IRIS DATASET:",

      X.std())

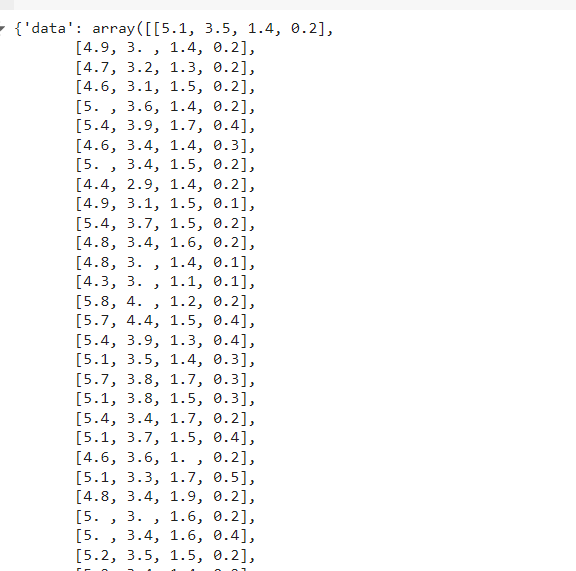
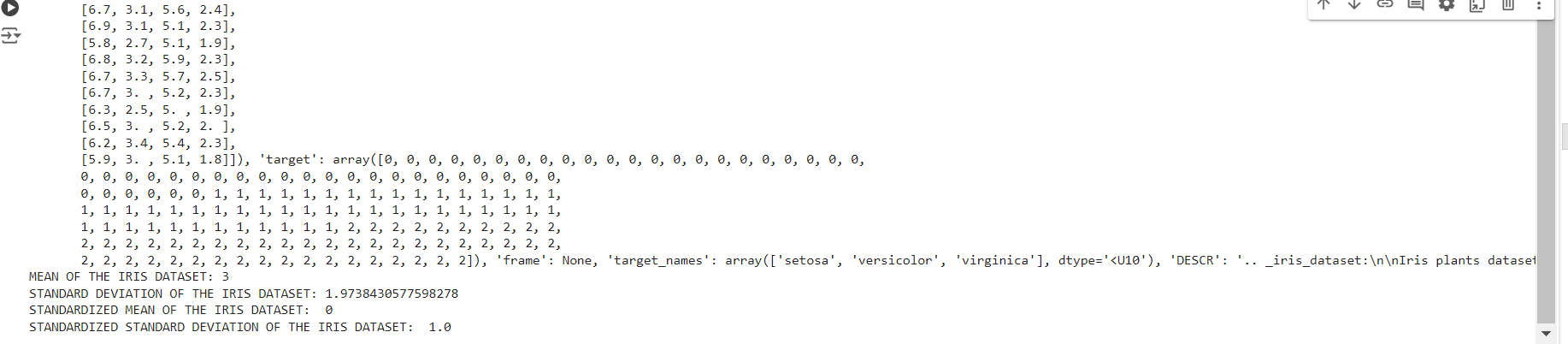
#standarize the data attributes

standardized\_X = preprocessing.scale(X)

print("STANDARDIZED MEAN OF THE IRIS DATASET: ", int(standardized\_X.mean()))

print("STANDARDIZED STANDARD DEVIATION OF THE IRIS DATASET: ", standardized\_X.std())

**OUTPUT:**

**** ****

**Section 2: Data Mining Techniques Run following algorithms on 2 real datasets and use appropriate evaluation measures to compute correctness of obtained patterns:**

**Ques 4. 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 %**

**CODE:**

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori

dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],

           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],

           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],

           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],

           ['corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

te = TransactionEncoder()

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

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

# minsup = 50% , minconf = 75%

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

print("\n\n Frequent Itemset 1 \n ", frequent\_itemsets)

from mlxtend.frequent\_patterns import association\_rules

a = association\_rules(frequent\_itemsets,

                      metric = "confidence", min\_threshold=0.75)

print("\n\n\n ASSOCIATION RULES : \n", a)

# minsup = 60% , minconf = 60%

frequent\_itemsets = apriori(df, min\_support=0.6,

                            use\_colnames=True)

print("\n\n\n Frequent Itemset 2 \n ", frequent\_itemsets)

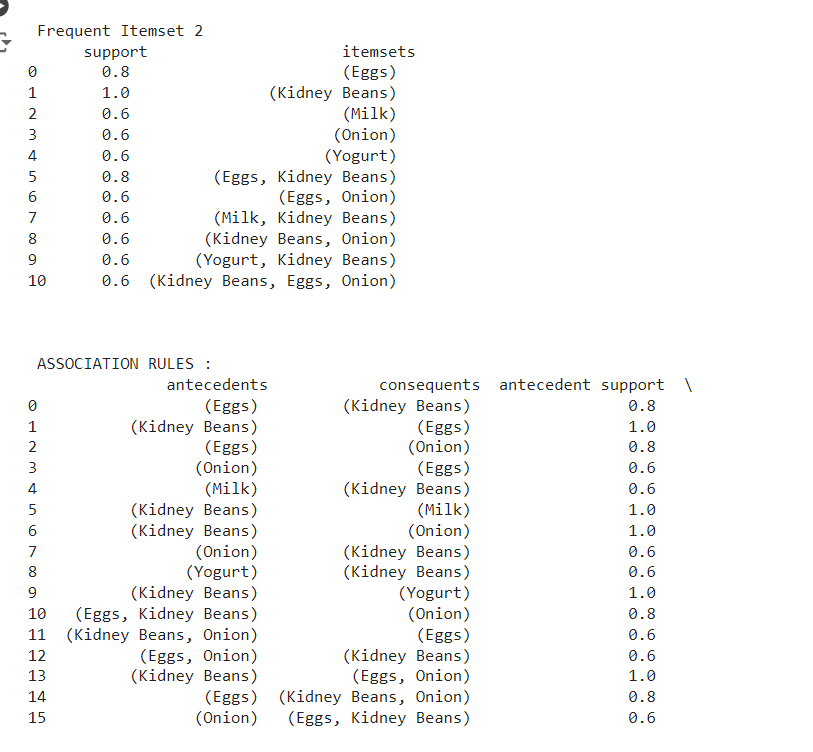
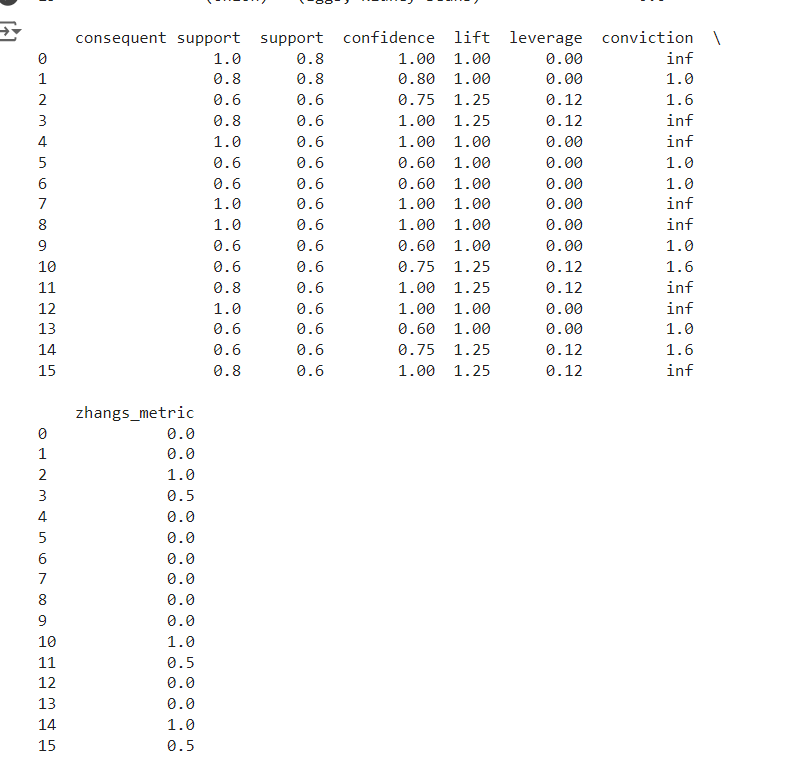
from mlxtend.frequent\_patterns import association\_rules

a = association\_rules(frequent\_itemsets,

                      metric = "confidence", min\_threshold=0.6)

print("\n\n\n ASSOCIATION RULES : \n", a)

**OUTPUT**

**** 

**Ques 5. 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.**

**CODE:**

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split,RepeatedKFold, KFold

from sklearn.metrics import confusion\_matrix

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import MinMaxScaler

ds = pd.read\_csv('iris.data.csv')

X = ds.values[:, :-1]

Y = ds.values[:, -1]

print(X.shape)

print(Y.shape)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=3)

#\*\*\*\*Naive Bayes Classifier

print("----------------NAIVE BAYES' CLASSIFIER \n")

NBclassifier = GaussianNB()

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

predictions = NBclassifier.predict(X\_test)

#print("Predicted response of NAIVE BAYES CLASSIFIER:\n", predictions)

#Accuracy Score

acc= accuracy\_score(Y\_test, predictions)

print("Accuracy score of NAIVE BAYES CLASSIFIER: ",acc)

# # Confusion Matrix

conmat= confusion\_matrix(Y\_test, predictions)

print("CONFUSION MATRIX of Y\_Test and Predictions: \n",

conmat)

#\*\*\*\*Decision Tree Classifier

print("\n----------------DECISION TREE CLASSIFIER \n")

DTclassifer= DecisionTreeClassifier()

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

predictions1 = DTclassifer.predict(X\_test)

#print("Predicted response of DECISION TREE CLASSIFIER:\n", predictions1)

#Accuracy Score

acc1= accuracy\_score(Y\_test, predictions1)

print("Accuracy score of DECISION TREE CLASSIFIER: ",acc1)

# # Confusion Matrix

conmat1= confusion\_matrix(Y\_test, predictions1)

print("CONFUSION MATRIX of Y\_Test and Predictions: \n",

conmat1)

#\*\*\*\*K- Nearest Neighbours

print("\n----------------K-NEAREST NEIGHBOUR \n")

# KNN model requires you to specify n\_neighbors,the number of points the classifier will look at to determine what class a new point belongs to

KNN\_model = KNeighborsClassifier(n\_neighbors=5)

KNN\_model.fit(X\_train, Y\_train)

predictions2 = KNN\_model.predict(X\_test)

#print("Predicted response of KNN: \n", predictions2)

#Accuracy Score

acc2= accuracy\_score(Y\_test, predictions1)

print("Accuracy score of KNN: ", acc2)

#Confusion Matrix

conmat2= confusion\_matrix(Y\_test, predictions2)

print("CONFUSION MATRIX of Y\_Test and Predictions: \n", conmat2)

print("\n----------------TRAINING SET SPLITTING USING HOLDOUT METHOD \n")

Val\_size = 0.25 # test size is how much rest is training

random\_seed = 3 # randomly choosing

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

Y, test\_size=Val\_size,

random\_state=random\_seed)

deciTree = DecisionTreeClassifier()

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

predictions3 = deciTree.predict(X\_test)

print("Accuracy on the Test Data when split is 0.25 using DECISION TREE")

print(accuracy\_score(Y\_test, predictions3))

Val\_size = 0.33 # test size is how much rest is

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

Y, test\_size=Val\_size,

random\_state=random\_seed)

deciTree = DecisionTreeClassifier()

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

predictions4 = deciTree.predict(X\_test)

print("Accuracy on the Test Data when split is 0.33 using DECISION TREE")

print(accuracy\_score(Y\_test, predictions4))

print("\n----------------TRAINING USING RANDOM FOREST CLASSIFIER \n")

# Performing training

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, Y,

test\_size = 0.3, random\_state = 100)

clfr= RandomForestClassifier(random\_state = 100)

clfr.fit(Xtrain, ytrain)

predictions5 = clfr.predict(Xtest)

print("Accuracy on the Test Data using RANDOM FOREST CLASSIFIER")

print(accuracy\_score(ytest, predictions5))

print("\n----------------TRAINING SET BY K-FOLD CROSS VALIDATION \n")

kf = KFold(n\_splits=5, random\_state=None, shuffle=False)

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

    X\_train1, X\_test1 = X[train\_index], X[test\_index]

    Y\_train1, Y\_test1 = Y[train\_index], Y[test\_index]

deciTree = DecisionTreeClassifier()

deciTree.fit(X\_train1, Y\_train1)

predictions = deciTree.predict(X\_test1)

print("Accuracy on the Test Data AFTER CROSS VALIDATION")

print(accuracy\_score(Y\_test1, predictions))

print("\n----------------SCALING TO A STANDARD FORMAT USING MINMAXSCALER \n")

#\*\*\*Scaling of data using minmaxscaler()-NORMALIZATION

scaler= MinMaxScaler()

df=pd.DataFrame(X, columns=['A', 'B', 'C', 'D'])

scalerfit= scaler.fit(df)

#print(scalerfit)

#print(scaler.data\_max\_)

scalertrans= scaler.transform(df)

print("MEAN OF THE IRIS DATASET: ", int(X.mean()))

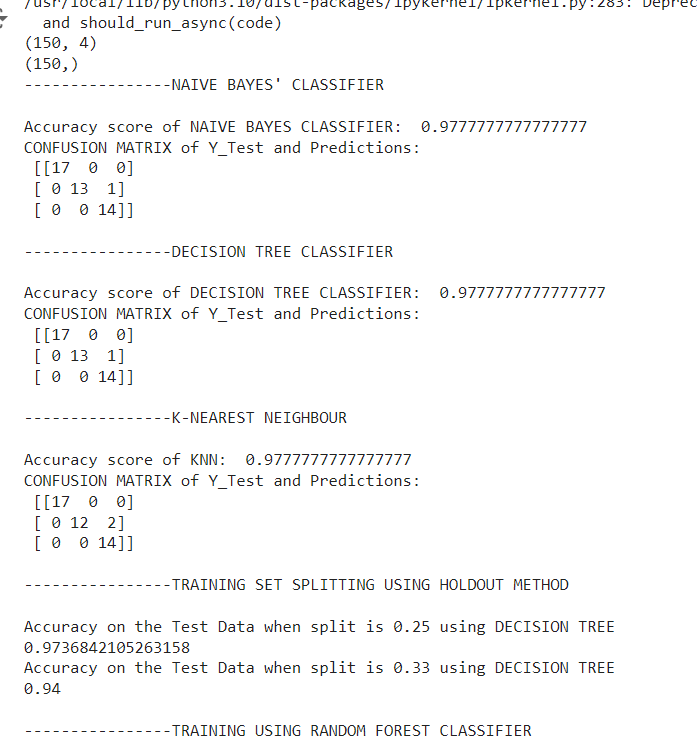
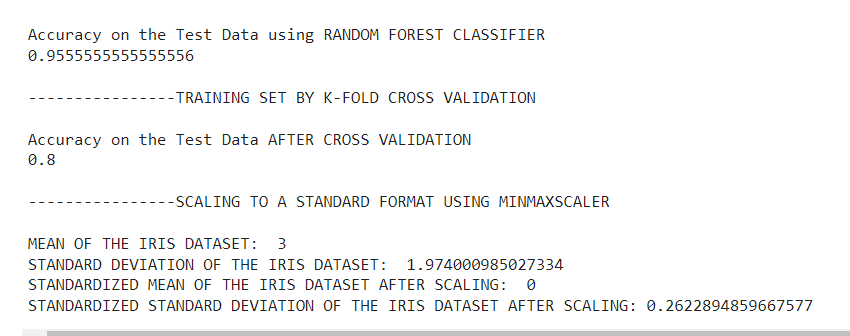
print("STANDARD DEVIATION OF THE IRIS DATASET: ",

X.std())

print("STANDARDIZED MEAN OF THE IRIS DATASET AFTER SCALING: ",int(scalertrans.mean()))

print("STANDARDIZED STANDARD DEVIATION OF THE IRIS DATASET AFTER SCALING:",scalertrans.std())

**OUTPUT:**

**** ****

**Ques 6. Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms.**

**CODE:**

from sklearn.cluster import KMeans

from sklearn import datasets

from sklearn.cluster import DBSCAN

from sklearn.cluster import AgglomerativeClustering

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

import pandas as pd

iris = datasets.load\_iris ()

x = iris.data[:, :4] #means we take 4 dimension in the feature space

#plot data distribution

plt.scatter(x[:, 0], x[:, 1], c="red", marker="o")

plt.title("ORIGINAL DATA SAMPLE")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.show()

#\*\*\*\*SIMPLE KMEANS\*\*\*\*

estimator = KMeans(n\_clusters=3) #construct a clusterer

estimator.fit(x) #clustering

label\_pred = estimator.labels\_ #get cluster labels

print("Number of samples per cluster in Simple K-Means Clustering:")

print(pd.Series(estimator.labels\_).value\_counts())

print(confusion\_matrix(iris.target, estimator.labels\_))

#Draw k-means results

x0 = x[label\_pred ==0]

x1 = x[label\_pred == 1]

x2 = x[label\_pred == 2]

plt.scatter(x0[:, 0], x0[:, 1], c="red", marker="o", label="Setosa")

plt.scatter(x1[:, 0], x1[:, 1], c="green", marker="\*", label="Versicolor")

plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker="+", label="Virginica")

plt.title("SIMPLE K-MEANS CLUSTERING")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.legend (loc=2)

plt.show()

#\*\*\*\*DBSCAN\*\*\*\*

dbscan = DBSCAN(eps=0.5, min\_samples=9)

dbscan.fit(x)

label\_pred = dbscan.labels\_

print("Number of samples per cluster in DBSCAN Clustering:")

print(pd.Series(dbscan.labels\_).value\_counts ())

print("DBSCAN Clustering Result:")

print(confusion\_matrix(iris.target, dbscan.labels\_))

#Draw k-means results

x0 = x[label\_pred == 0]

x1 = x[label\_pred == 1]

x2 = x[label\_pred == 2]

plt.scatter(x0[:, 0], x0[:, 1], c="red", marker="o", label="Setosa")

plt.scatter(x1[:, 0], x1[:, 1], c="green", marker="\*", label="Versicolor")

plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker="+", label="Virginica")

plt.title("DBSCAN")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.legend (loc=2)

plt.show()

#\*\*\*\*HIREECHICAL \*\*\*\*

irisdata = iris.data

clustering = AgglomerativeClustering(linkage="ward", n\_clusters=3)

res = clustering.fit (irisdata)

print("Number of samples per cluster in Hirerachical Clustering:")

print(pd.Series(clustering.labels\_).value\_counts())

print("Hirerachical Clustering Result:")

print(confusion\_matrix(iris.target, clustering.labels\_))

plt.figure()

d0 = irisdata [clustering.labels\_ == 0]

plt.plot(d0[:, 1], d0[:,1], "r.")

d1 = irisdata [clustering.labels\_ == 1]

plt.plot (d1[:,0], d1[:,1], "go")

d2 = irisdata [clustering.labels\_ ==2]

plt.plot (d2[:, 0], d2[:,1], "b \*")

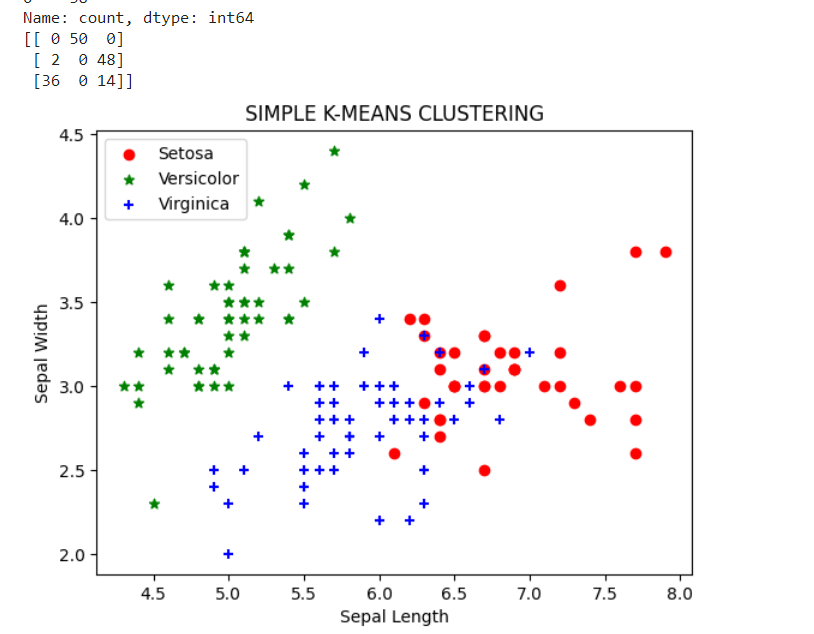
plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.title("HIRERCHICAL (AGNES) CLUSTERING")

plt.show()

**OUTPUT:**

**** 