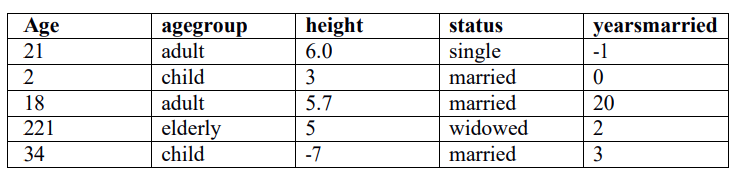
DATA MINING

Practical File

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



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

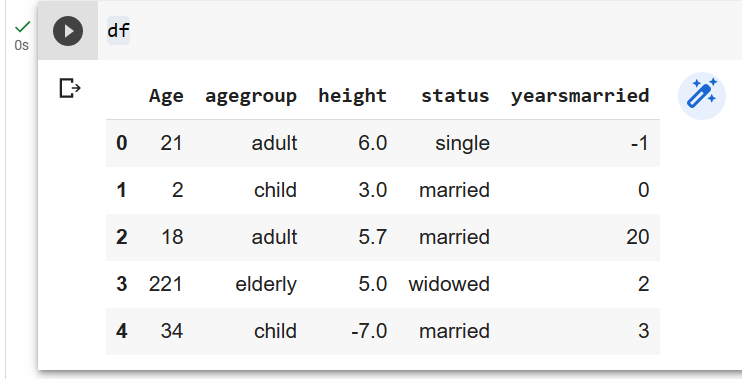
1. **Check whether ruleset E is violated by the data in the file people.txt.**
2. **Summarize the results obtained in part (iii)**
3. **Visualize the results obtained in part (iii)**

import pandas as pd

import numpy as np

df = pd.read\_table("people.txt")

df



isvalidAge = (df["Age"]>=0)  &  (df["Age"]<=150)

s1 = isvalidAge.shape[0]

invalid1 = s1 - isvalidAge.sum()

invalid1

isvalidAgeMarried=(df["Age"] >= df["yearsmarried"])

invalid2 = isvalidAgeMarried.count()- isvalidAgeMarried.sum()

invalid2

isvalidStatus = (df["status"] == "single") | (df["status"]=="married") |(df["status"] == "widowed")

invalid3 =isvalidStatus.count() -  isvalidStatus.sum()

invalid3

isvalidAgegroup = (df["Age"]<18) & (df['agegroup'] == 'child') | ((df["Age"]>=18) & (df["Age"]<65)) &(df['agegroup'] =='adult') |  (df["Age"]>65) &( df['agegroup'] == 'elderly')

invalid4 = isvalidAgegroup.count() -  isvalidAgegroup.sum()

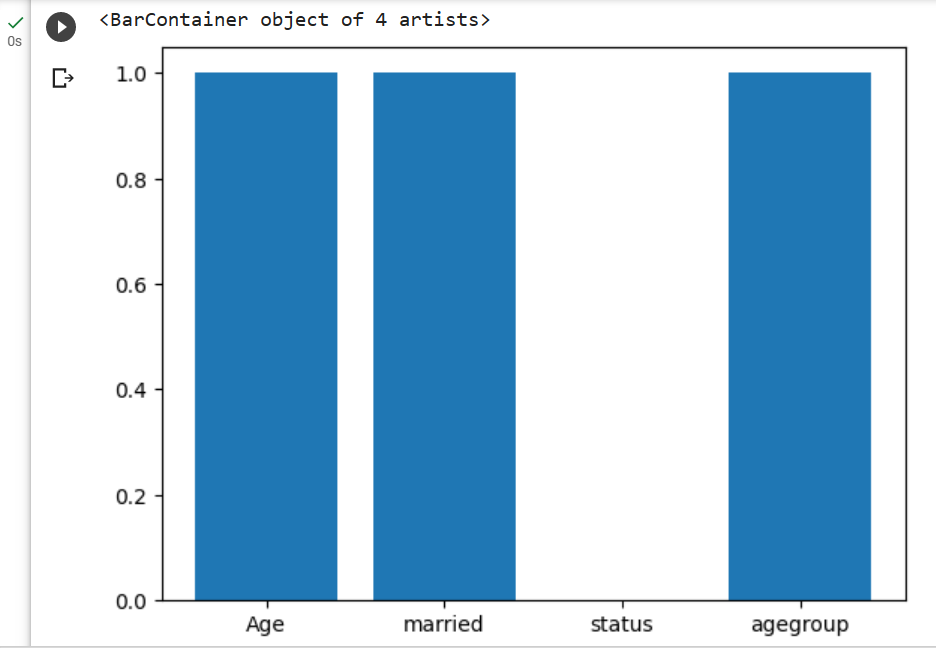
invalid4

import matplotlib.pyplot as plt

x=["Age","married","status","agegroup"]

y=[invalid1,invalid2,invalid3,invalid4]

plt.bar(x,y)



def ruleset(df):

    df["Rule1"]=df["Age"].apply(lambda x: True if x>0 and x<150 else False)

    df["Rule2"]=df.apply(lambda x:True if x.Age>x.yearsmarried else False,axis=1)

    df["Rule3"]=df.apply(lambda x:True if (x.status=="single")|(x.status=="married" and x.Age>18)|(x.status=="widowed" and x.Age>18) else False,axis=1)

    df['Rule4']=df.apply(lambda x: True if (x.Age<18 and x.agegroup=='child')|((x.Age>=18 and x.Age<65) and x.agegroup=='adult' ) | (x.Age>65 and x.agegroup == 'elderly') else False, axis=1)

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

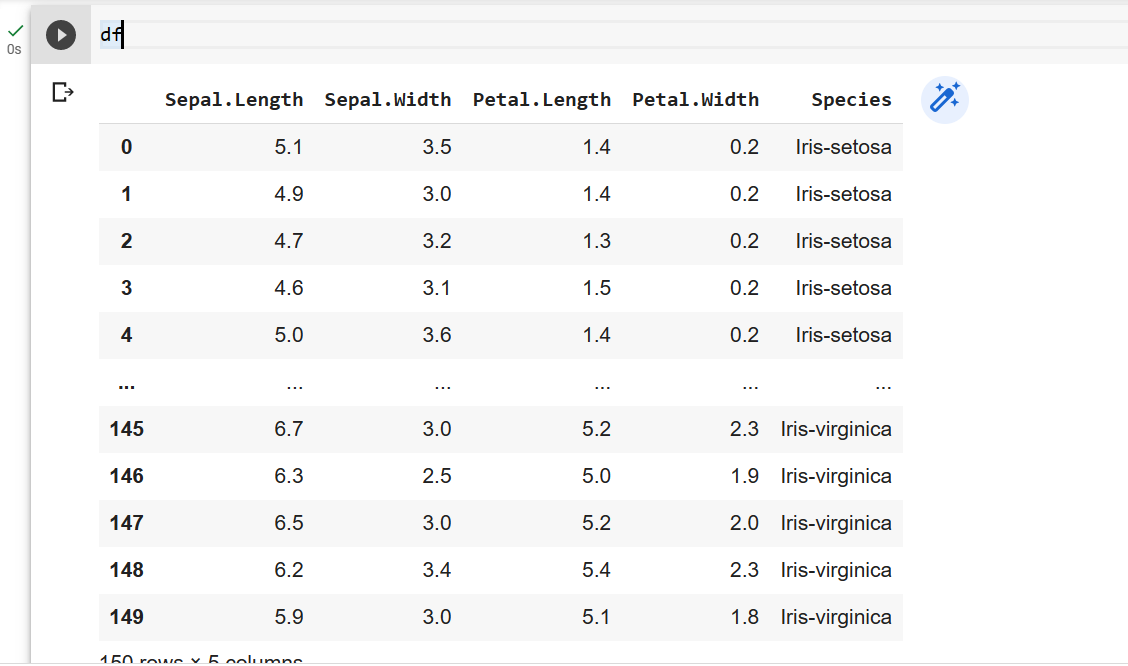
1. **Calculate the number and percentage of observations that are complete.**
2. **Replace all the special values in data with NA.**
3. **Define these rules in a separate text file and read them. (Use editfile function in R (package editrules). Use similar function in Python). Print the resulting constraint 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.**
4. **Determine how often each rule is broken (violatedEdits). Also summarize and plot the result.**
5. **Find outliers in sepal length using boxplot and boxplot.stats**

import pandas as pd

import numpy as np

df = pd.read\_csv('dirty\_iris.txt')

df



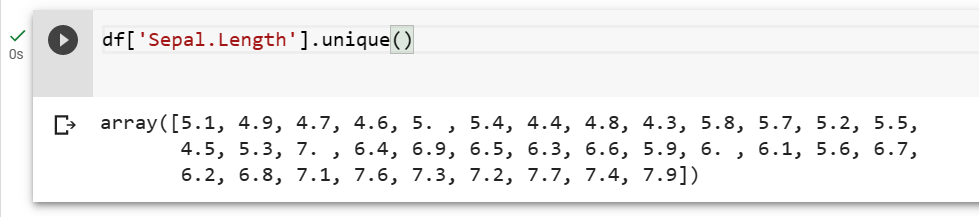
df1 = (df.isna().any(axis=1)==0).sum()

df1

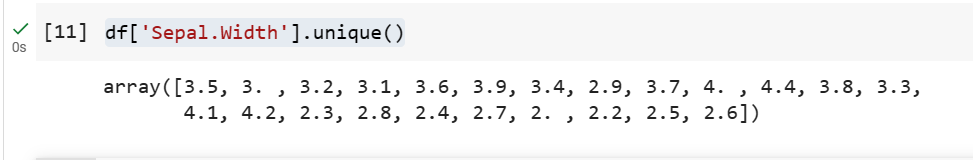
df3 = (df1/(len(df)))\*100

print("Percentage of No of observation", df3)

df['Sepal.Length'].unique()

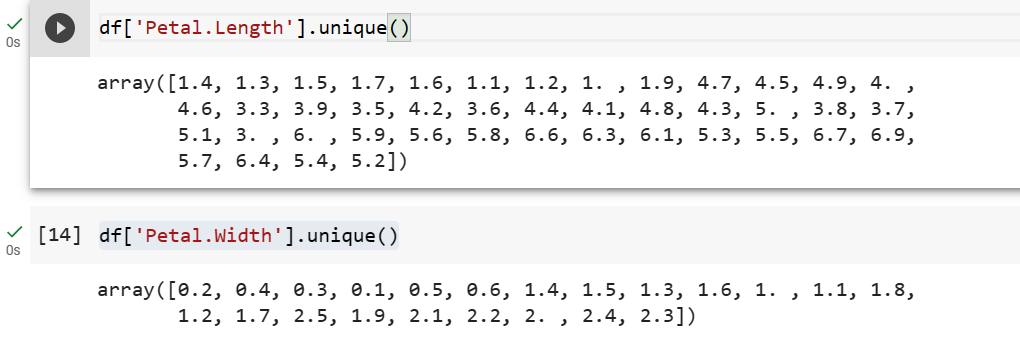


df['Sepal.Width'].unique()



df['Petal.Length'].unique()

df['Petal.Width'].unique()



df['Petal.Width'].replace(np.inf,0,inplace=True)

df['Petal.Width'].unique()

isValidSpecies = (df["Species"]=="setosa") | (df["Species"]=="virginica") | (df["Species"]=="versicolor")

a=isValidSpecies.count()-isValidSpecies.sum()

isValidSpecies = (df["Sepal.Length"]>0) | (df["Petal.Length"]>0) | (df["Petal.Width"]>0) | (df["Sepal.Length"]>0)

b=isValidSpecies.count() -isValidSpecies.sum()

isValidLength = (df["Petal.Length"]) >(df["Petal.Width"].astype("float64")\*2)

c=isValidLength.count() -isValidLength.sum()

isValidSepLen = df["Sepal.Length"]<30

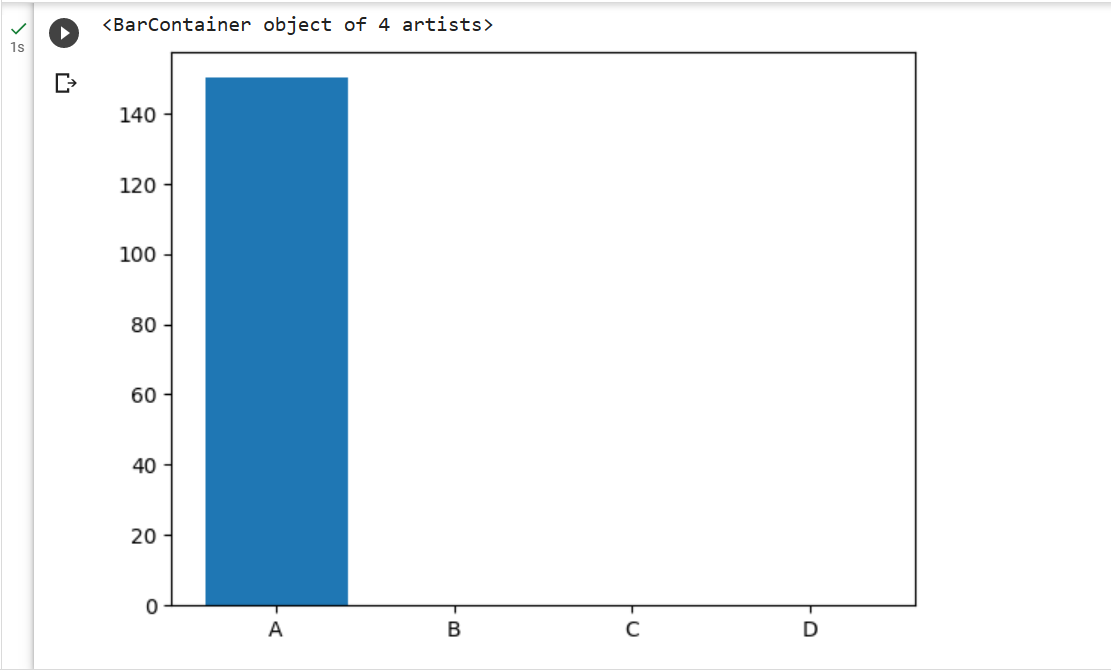
d=isValidSepLen.count() -isValidSepLen.sum()

import matplotlib.pyplot as plt

x=["A","B","C","D"]

y=[a,b,c,d]

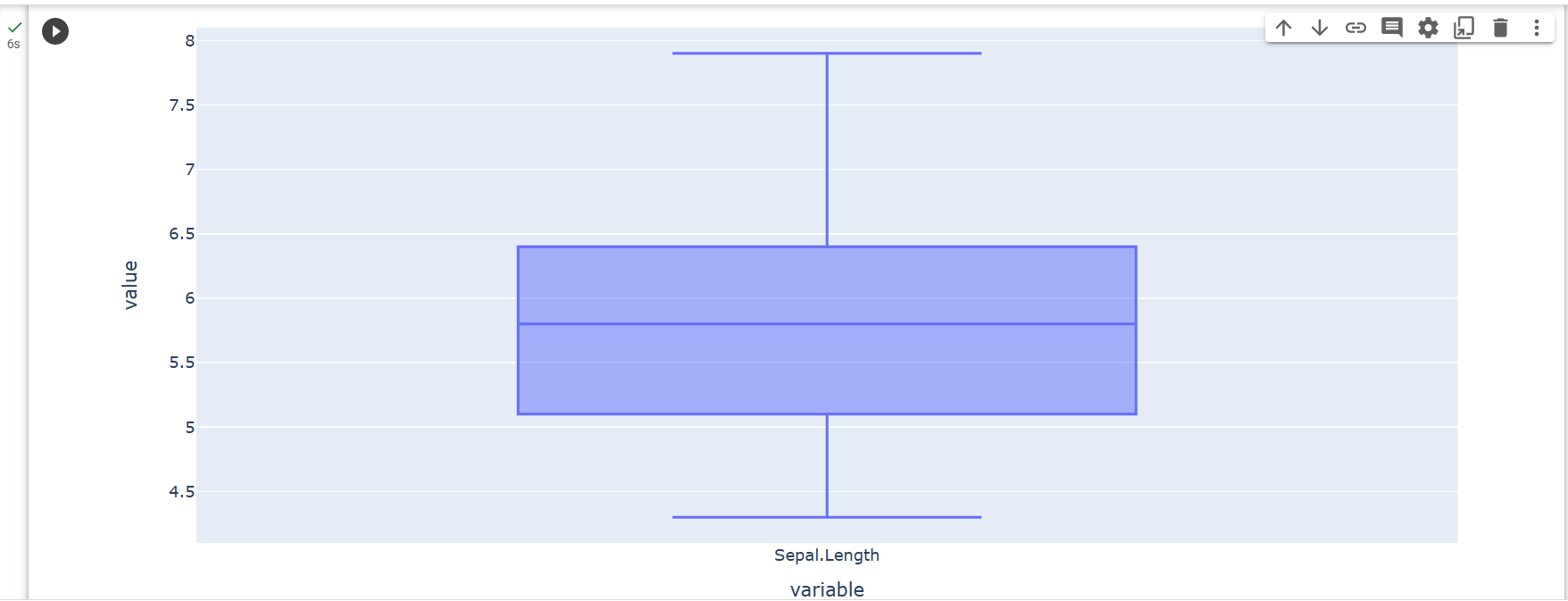
plt.bar(x,y)



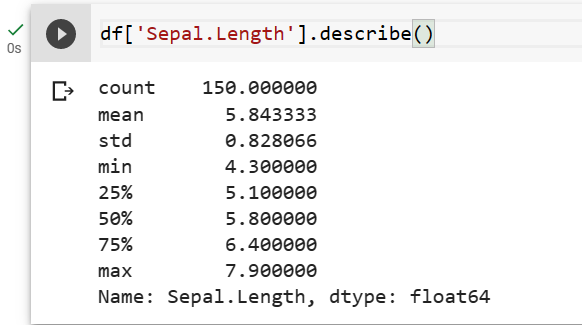
isLonger = df["Sepal.Length"]>df["Petal.Length"]

import plotly.express as px

px.box(df["Sepal.Length"])



df['Sepal.Length'].describe()



**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.**

**Iris Dataset:**

from sklearn.datasets import load\_iris

import pandas as pd

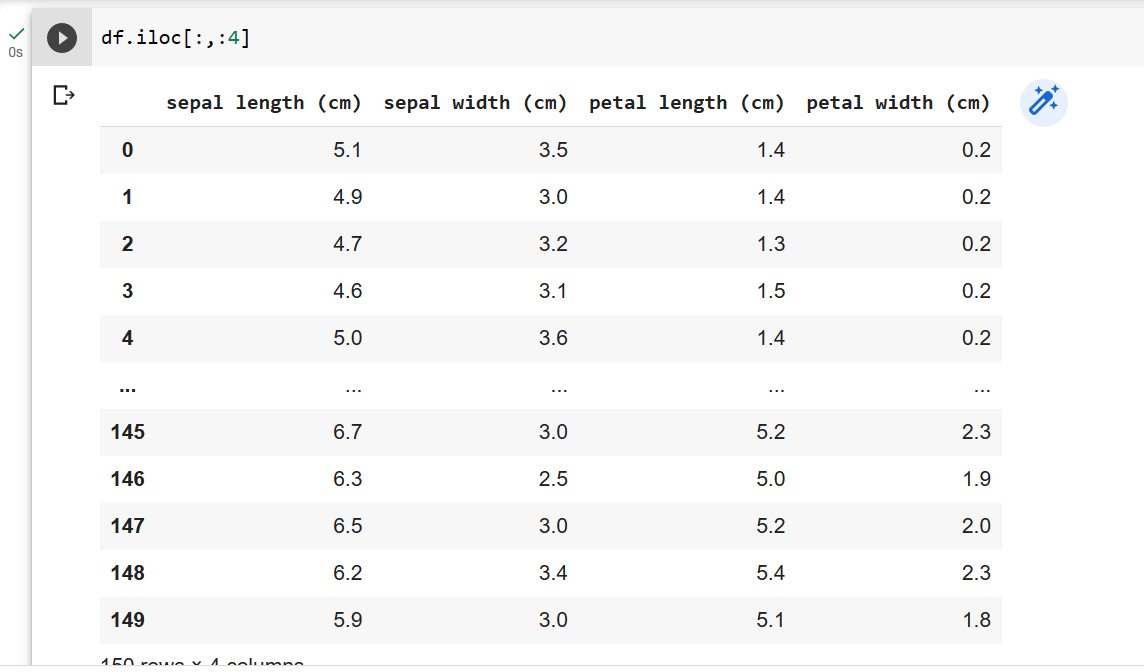
iris = load\_iris()

df = pd.DataFrame(iris.data,columns=iris.feature\_names)

df['Class'] = iris.target

df.mean()

df.iloc[:,:4]



df.std()

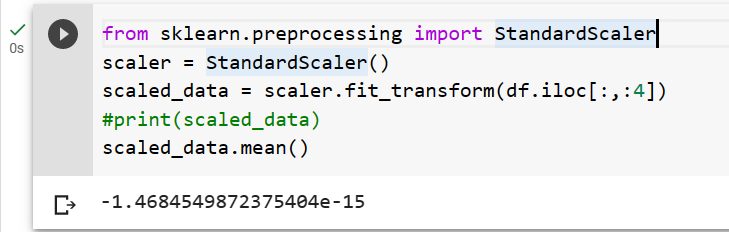
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(df.iloc[:,:4])

#print(scaled\_data)

scaled\_data.mean()



scaled\_data.std()

**Wine Data det:**

from sklearn.datasets import load\_wine

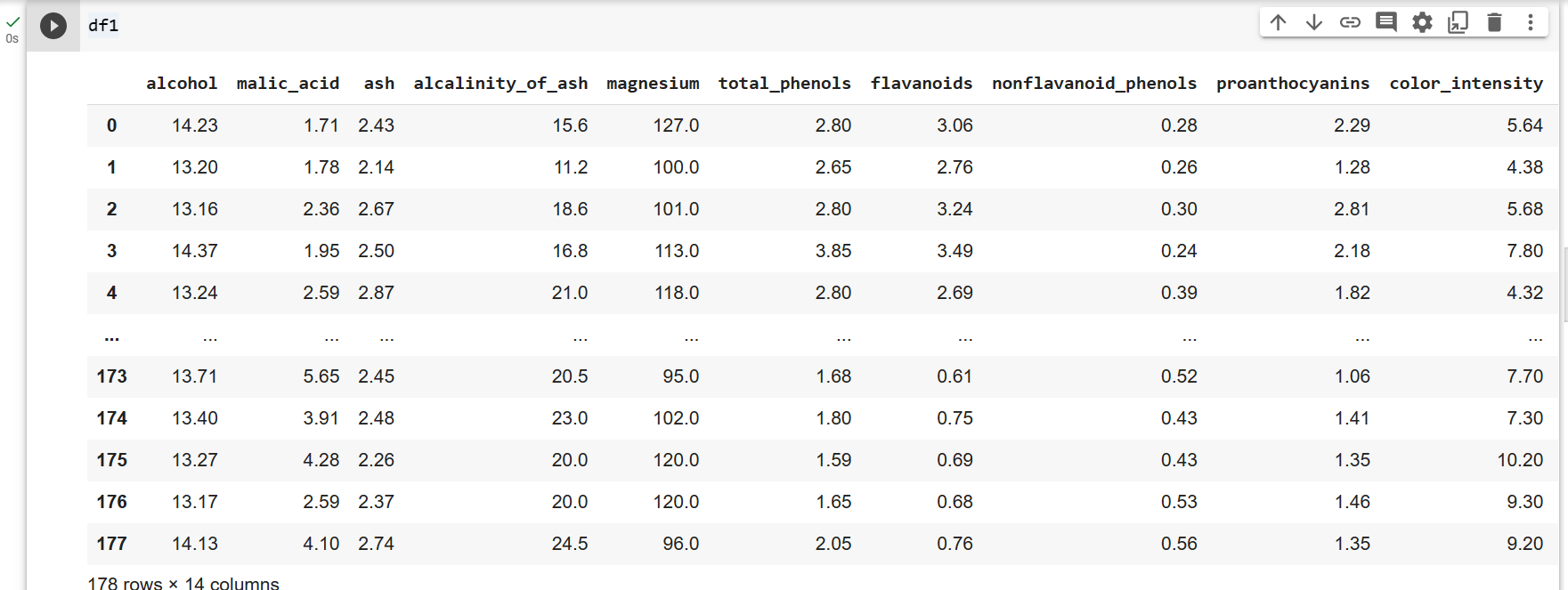
import pandas as pd

wine = load\_wine()

df1 = pd.DataFrame(wine.data,columns=wine.feature\_names)

df1['Class'] = wine.target

df1



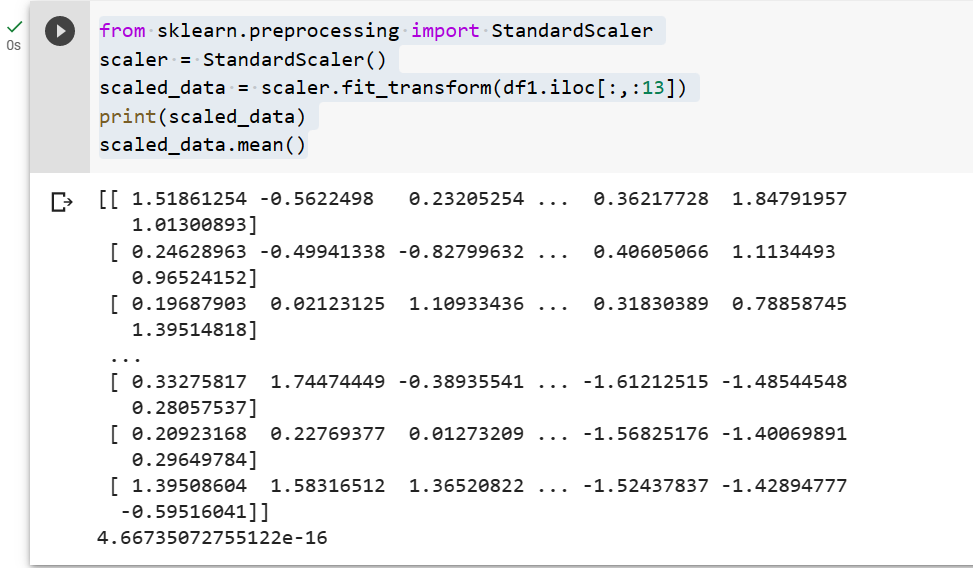
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(df1.iloc[:,:13])

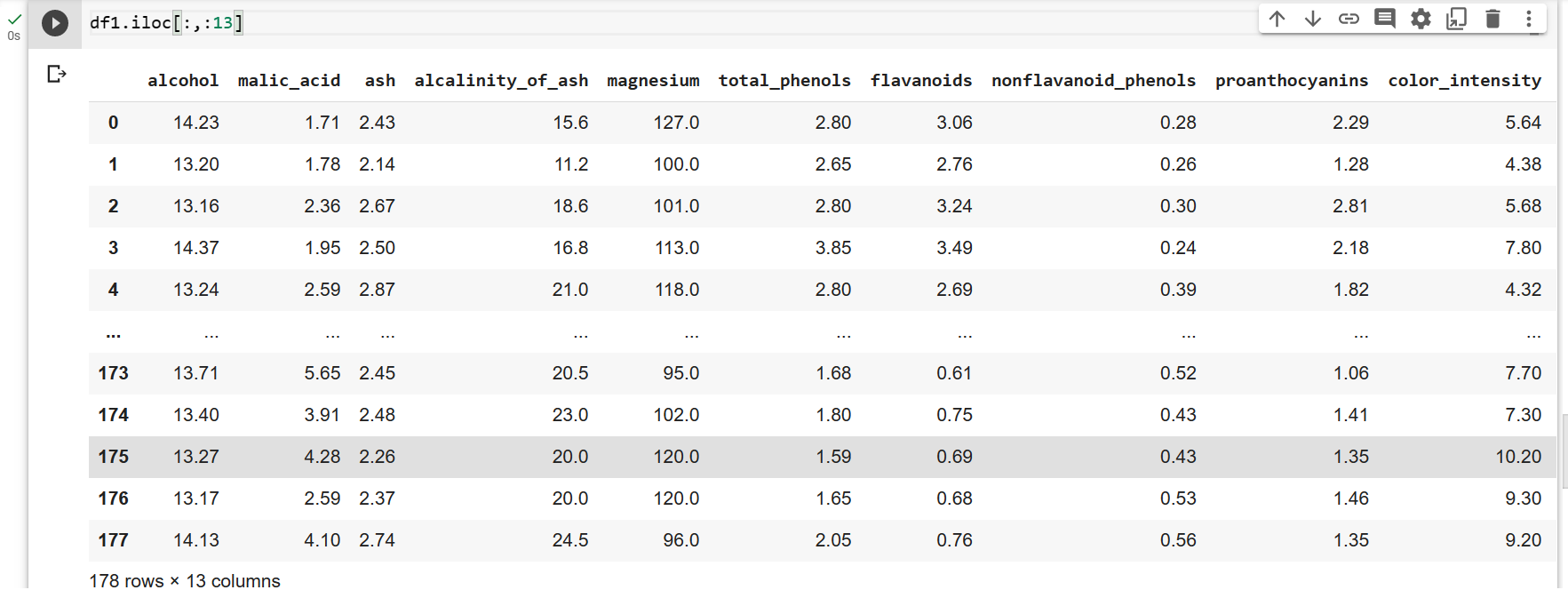
print(scaled\_data)

scaled\_data.mean()



scaled\_data.std()

df1.iloc[:,:13]



**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 %.**

**Groceries dataset:**

!pip install efficient-apriori

from efficient\_apriori import apriori

import pandas as pd

df=pd.read\_csv("C:/Users/student/Downloads/groceriesDataset.csv")

df

transactions=[["milk","diaper","Tea"],

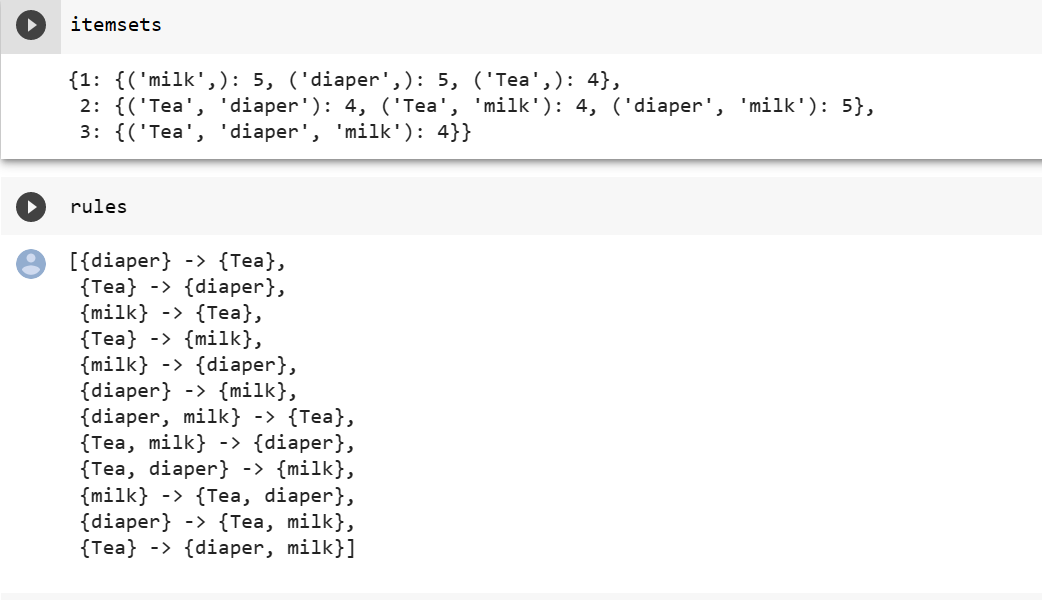
             ["milk","diaper","Tea","bread","butter"],

             ["milk","diaper","Tea"],

             ["butter","milk","diaper","bread"],

             ["milk","diaper","bread","butter","Tea"]]

itemsets,rules =apriori(transactions,min\_support=0.75, min\_confidence=0.75)



df.columns

li=[]

for i in range(len(df)):

    t=[]

    for j in range(df["Item(s)"][i]):

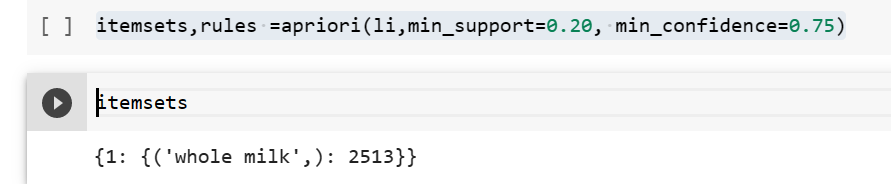
        t.append(df[df.columns[j+1]][i])

    li.append(t)

li

itemsets,rules =apriori(li,min\_support=0.5, min\_confidence=0.75)

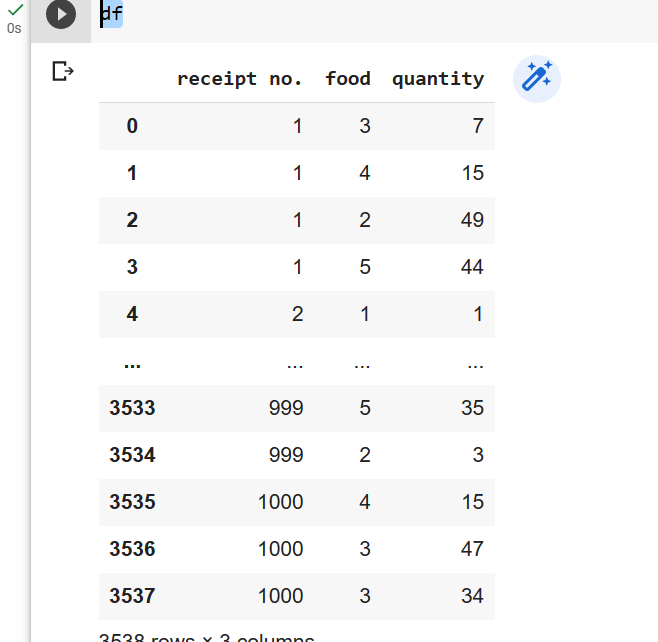
itemsets,rules =apriori(li,min\_support=0.20, min\_confidence=0.75)



**Bakery data set:**

df=pd.read\_csv("1000i.csv",header=None,names=["receipt no.","food","quantity"])

df



dict={

    1:"milk",2:"sugar",3:"chocolate",4:"apples",5:"curd"

}

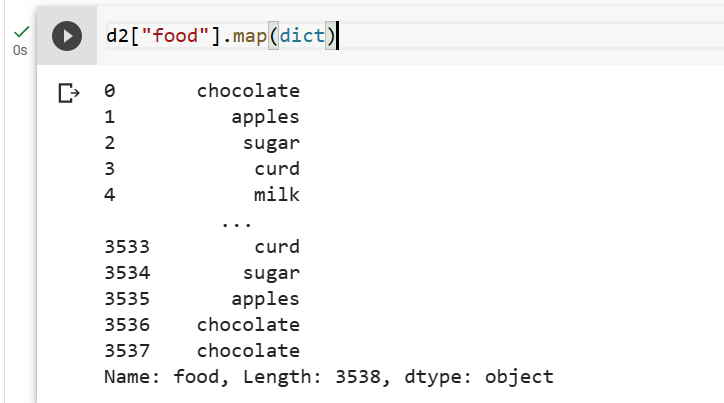
d1=pd.DataFrame(df)

d1

d2=pd.DataFrame(df)

d2

d2["food"].map(dict)

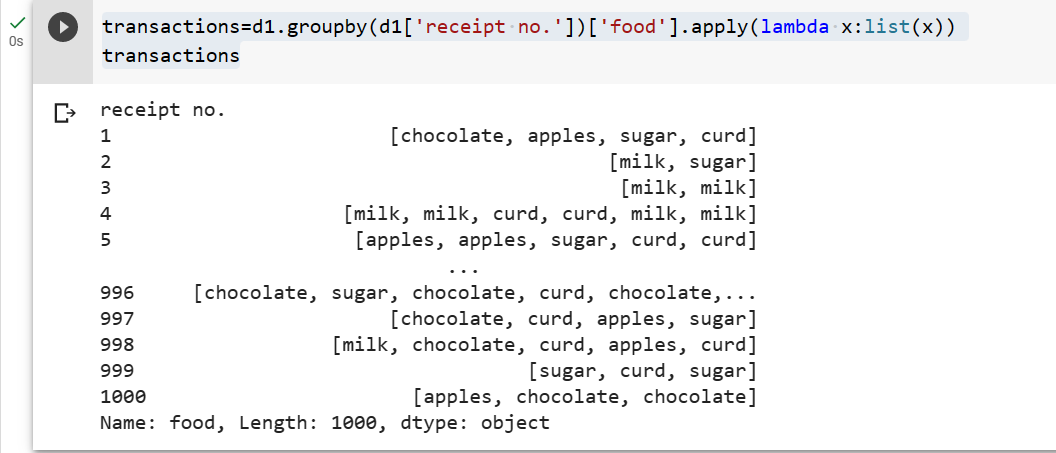


d1['food'] = d1['food'].replace(dict)

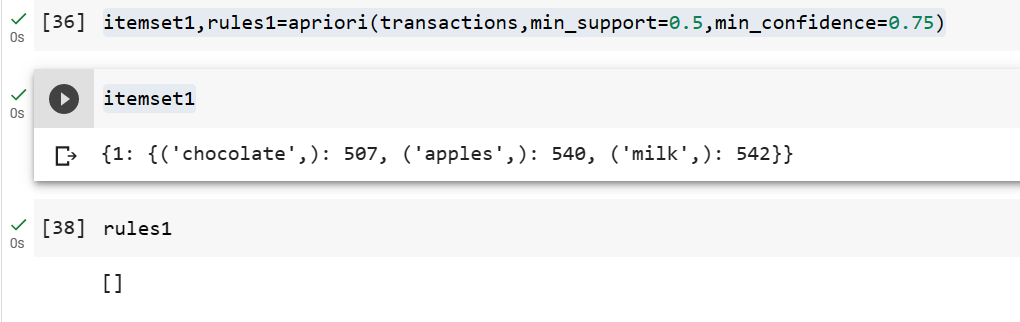
d1

transactions=d1.groupby(d1['receipt no.'])['food'].apply(lambda x:list(x))

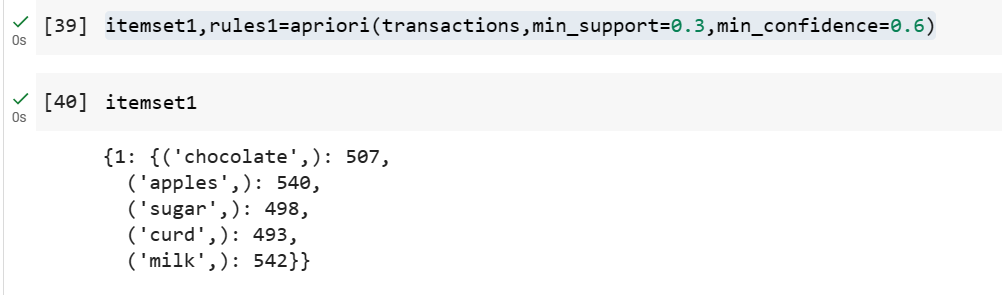
transactions



itemset1,rules1=apriori(transactions,min\_support=0.5,min\_confidence=0.75)



itemset1,rules1=apriori(transactions,min\_support=0.3,min\_confidence=0.6)



**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.**

!pip install sklearn

from matplotlib import pyplot as plt

from sklearn.datasets import load\_breast\_cancer

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

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

from sklearn.metrics import confusion\_matrix,precision\_score,recall\_score,f1\_score,ConfusionMatrixDisplay

svm = load\_breast\_cancer()

x = svm.data

y = svm.target

print(x.shape)

print(y.shape)

k\_size = 0.25

r\_seed = 100

X\_train,X\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=k\_size,random\_state=r\_seed)

print("Shape of X\_train" ,X\_train.shape)

print("Shape of y\_train" ,y\_train.shape)

print("Shape of X\_train" ,X\_test.shape)

print("Shape of y\_train" ,y\_test.shape)

Decision Tree of Hold Out

deci\_tree =DecisionTreeClassifier(criterion ="entropy") # By default gini

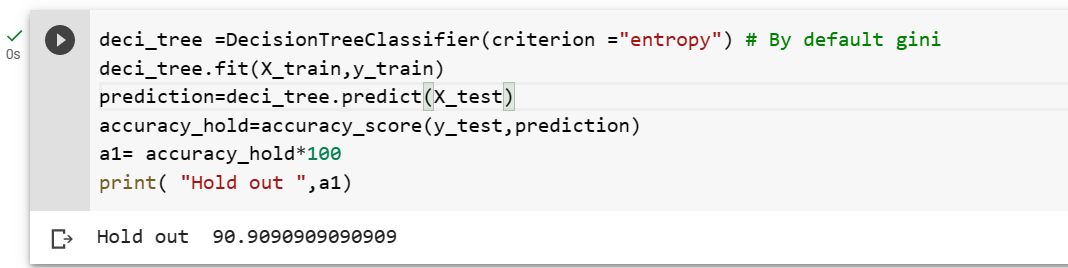
deci\_tree.fit(X\_train,y\_train)

prediction=deci\_tree.predict(X\_test)

accuracy\_hold=accuracy\_score(y\_test,prediction)

a1= accuracy\_hold\*100

print( "Hold out ",a1)

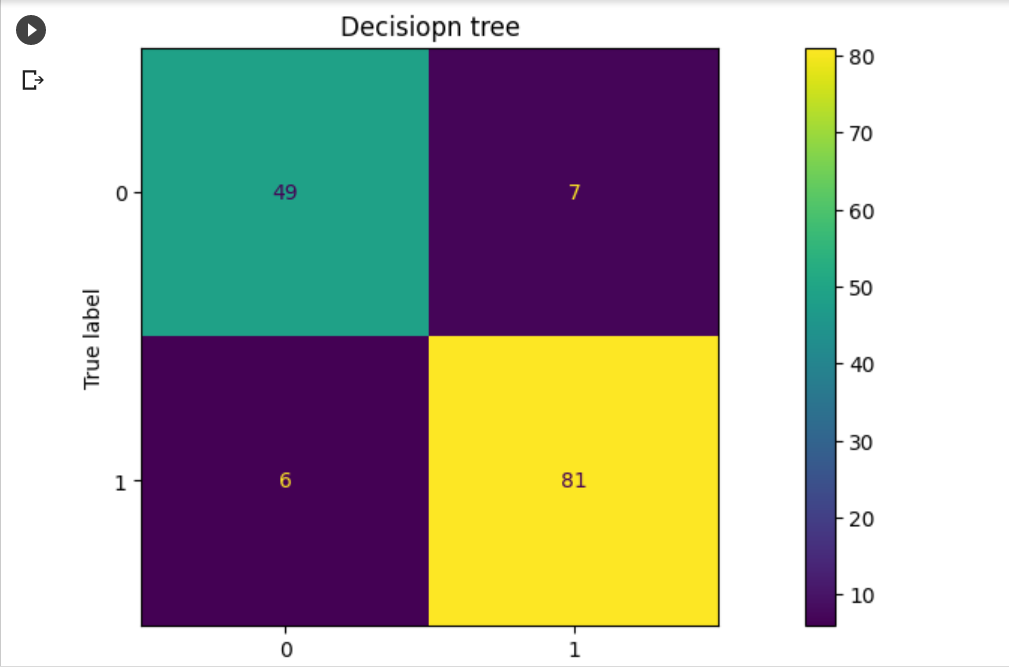


fig,(ax1)=plt.subplots()

fig.set\_size\_inches(15,5)

ax1.set\_title("Decisiopn tree")

ConfusionMatrixDisplay.from\_estimator(deci\_tree,X\_test,y\_test,ax=ax1)



# KNN

#KNN (Nearst Neighbour)

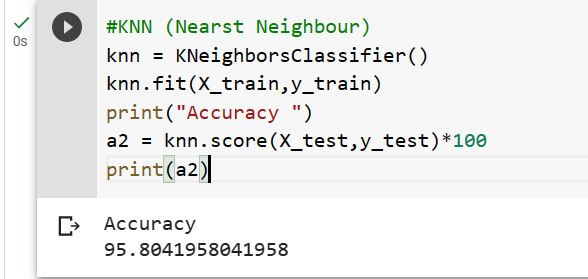
knn = KNeighborsClassifier()

knn.fit(X\_train,y\_train)

print("Accuracy ")

a2 = knn.score(X\_test,y\_test)\*100

print(a2)



# Naive bayes

from sklearn.naive\_bayes import GaussianNB

#fitting the model

nb=GaussianNB()

nb.fit(X\_train,y\_train)

prediction\_nb =nb.predict(X\_test)

print("Accuracy of Test Data")

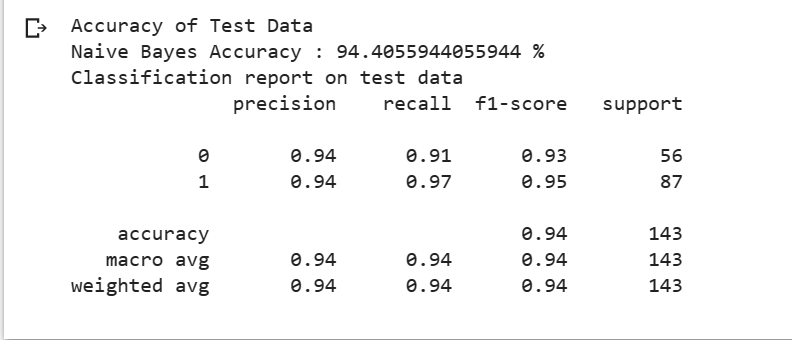
nb\_score=accuracy\_score(y\_test,prediction\_nb)

a3 = nb\_score\*100

print("Naive Bayes Accuracy :",a3,"%")

print("Classification report on test data")

print(classification\_report(y\_test,prediction\_nb))



import numpy as np

k=20

dta = []

for i in range(k):

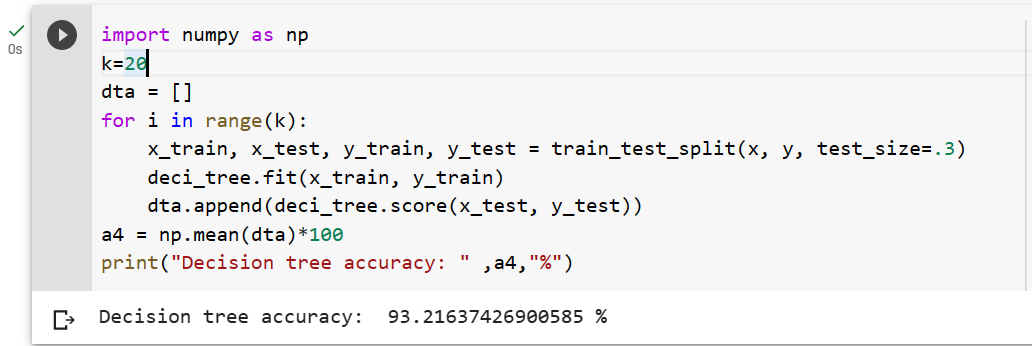
    x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=.3)

    deci\_tree.fit(x\_train, y\_train)

    dta.append(deci\_tree.score(x\_test, y\_test))

a4 = np.mean(dta)\*100

print("Decision tree accuracy: " ,a4,"%")



# Cross validation

from sklearn.model\_selection import cross\_val\_score

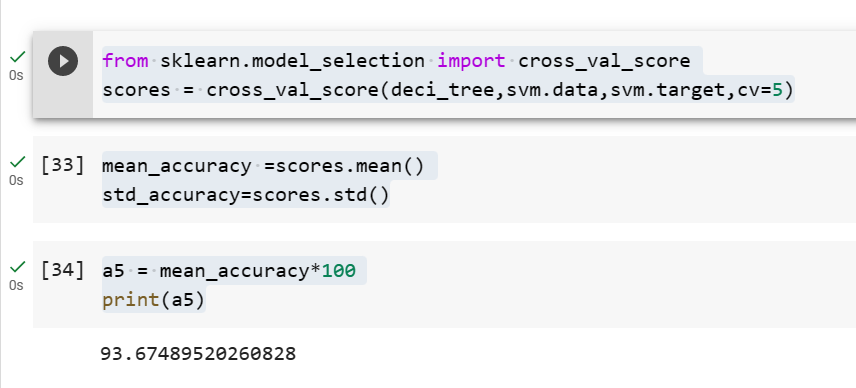
scores = cross\_val\_score(deci\_tree,svm.data,svm.target,cv=5)

mean\_accuracy =scores.mean()

std\_accuracy=scores.std()

a5 = mean\_accuracy\*100

print(a5)

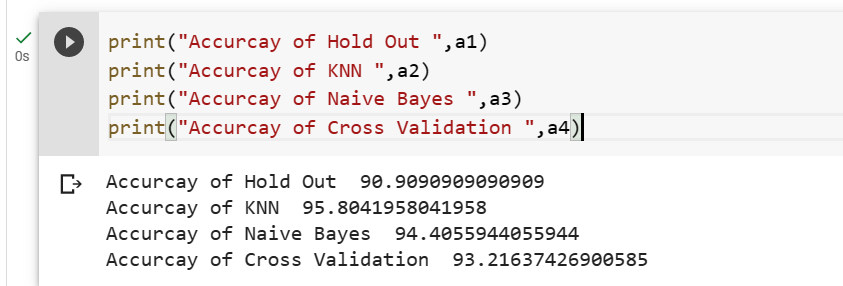


print("Accurcay of Hold Out ",a1)

print("Accurcay of KNN ",a2)

print("Accurcay of Naive Bayes ",a3)

print("Accurcay of Cross Validation ",a4)



**Using Iris Data set:**

!pip install sklearn

from matplotlib import pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

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

from sklearn.metrics import confusion\_matrix,precision\_score,recall\_score,f1\_score,ConfusionMatrixDisplay

iris = load\_iris()

x=iris.data

y=iris.target

print(x.shape)

print(y.shape)

k\_size=0.25

r\_seed=100

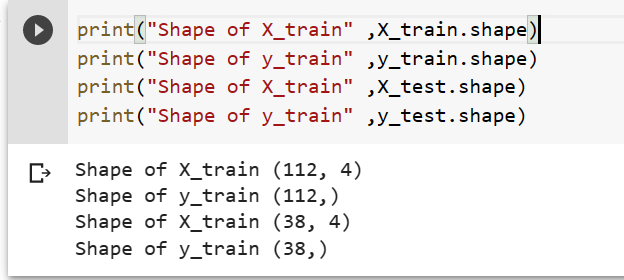
X\_train,X\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=k\_size,random\_state=r\_seed)

print("Shape of X\_train" ,X\_train.shape)

print("Shape of y\_train" ,y\_train.shape)

print("Shape of X\_train" ,X\_test.shape)

print("Shape of y\_train" ,y\_test.shape)



# Decision Tree of Hold Out

deci\_tree =DecisionTreeClassifier(criterion ="entropy") # By default gini

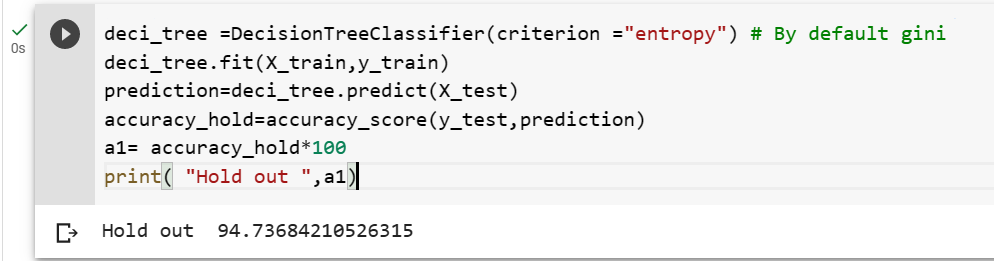
deci\_tree.fit(X\_train,y\_train)

prediction=deci\_tree.predict(X\_test)

accuracy\_hold=accuracy\_score(y\_test,prediction)

a1= accuracy\_hold\*100

print( "Hold out ",a1)

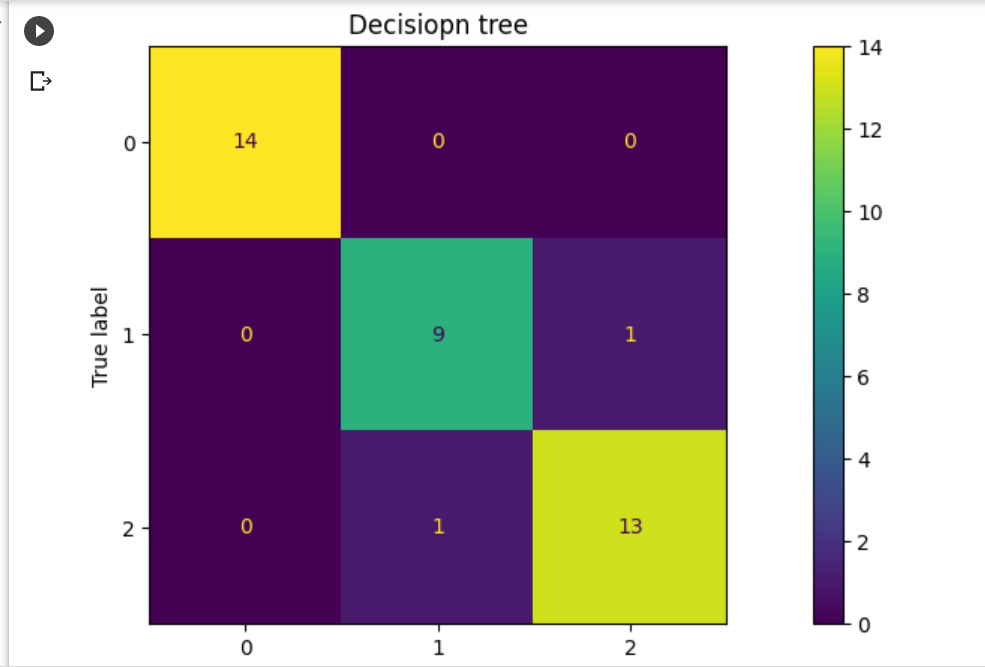


fig,(ax1)=plt.subplots()

fig.set\_size\_inches(15,5)

ax1.set\_title("Decisiopn tree")

ConfusionMatrixDisplay.from\_estimator(deci\_tree,X\_test,y\_test,ax=ax1)



# KNN

#KNN (Nearst Neighbour)

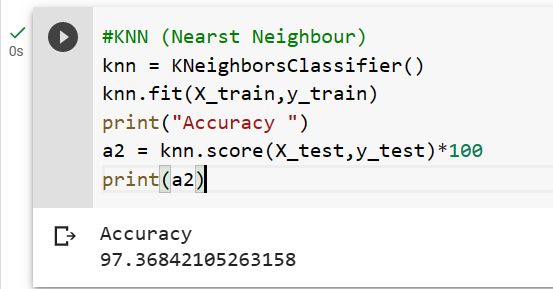
knn = KNeighborsClassifier()

knn.fit(X\_train,y\_train)

print("Accuracy ")

a2 = knn.score(X\_test,y\_test)\*100

print(a2)



# Naive bayes

from sklearn.naive\_bayes import GaussianNB

#fitting the model

nb=GaussianNB()

nb.fit(X\_train,y\_train)

prediction\_nb =nb.predict(X\_test)

print("Accuracy of Test Data")

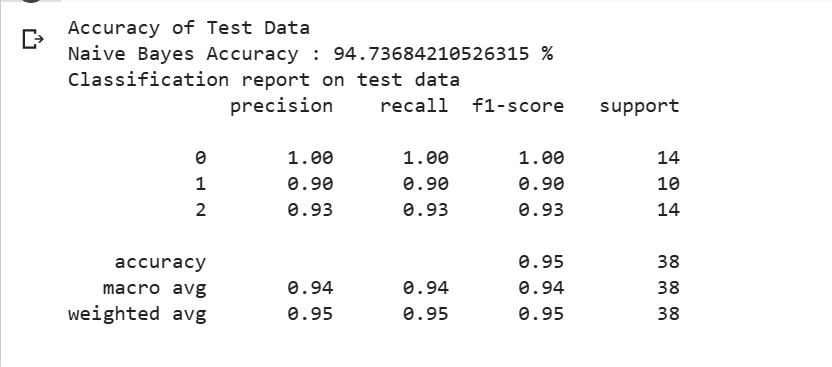
nb\_score=accuracy\_score(y\_test,prediction\_nb)

a3 = nb\_score\*100

print("Naive Bayes Accuracy :",a3,"%")

print("Classification report on test data")

print(classification\_report(y\_test,prediction\_nb))



import numpy as np

k=20

dta = []

for i in range(k):

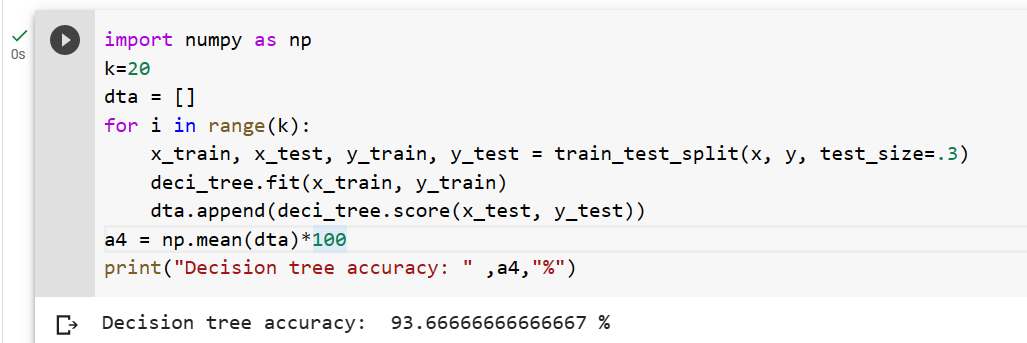
    x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=.3)

    deci\_tree.fit(x\_train, y\_train)

    dta.append(deci\_tree.score(x\_test, y\_test))

a4 = np.mean(dta)\*100

print("Decision tree accuracy: " ,a4,"%")



# Cross validation

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(deci\_tree,iris.data,iris.target,cv=5)

mean\_accuracy =scores.mean()

std\_accuracy=scores.std()

a5 = mean\_accuracy\*100

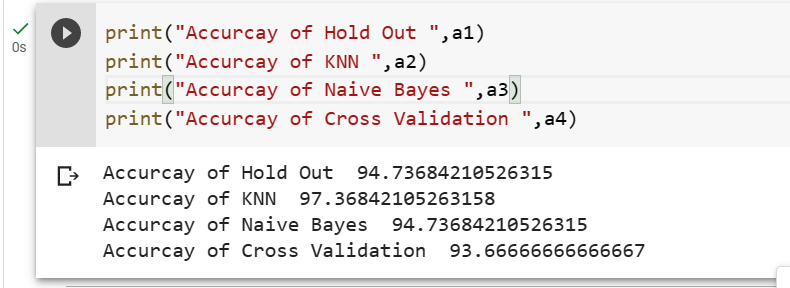
print(a5)

print("Accurcay of Hold Out ",a1)

print("Accurcay of KNN ",a2)

print("Accurcay of Naive Bayes ",a3)

print("Accurcay of Cross Validation ",a4)



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

import numpy as np

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score

from scipy.cluster.hierarchy import dendrogram, linkage

# Load breast\_cancer dataset

brct = load\_breast\_cancer()

X = brct.data

y = brct.target

# Simple KMeans

kmeans = KMeans(n\_clusters=3, init='k-means++', n\_init=10, max\_iter=300)

y\_kmeans = kmeans.fit\_predict(X)

# DBSCAN

dbscan = DBSCAN(eps=0.8, min\_samples=5)

y\_dbscan = dbscan.fit\_predict(X)

# Hierarchical Clustering

hc = AgglomerativeClustering(n\_clusters=3, linkage='ward')

y\_hc = hc.fit\_predict(X)

# Compare performance using accuracy score

kmeans\_score = accuracy\_score(y, y\_kmeans)

dbscan\_score = accuracy\_score(y, y\_dbscan)

hc\_score = accuracy\_score(y, y\_hc)

print("Accuracy Score - KMeans:", kmeans\_score\*100)

print("Accuracy Score - DBSCAN:", dbscan\_score\*100)

print("Accuracy Score - Hierarchical Clustering:", hc\_score\*100)

# Visualize the clustering

plt.subplot(131)

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis')

plt.title("KMeans")

plt.subplot(132)

plt.scatter(X[:, 0], X[:, 1], c=y\_dbscan, s=50, cmap='viridis')

plt.title("DBSCAN")

plt.subplot(133)

plt.scatter(X[:, 0], X[:, 1], c=y\_hc, s=50, cmap='viridis')

plt.title("Hierarchical Clustering")

# plt.show()

linked = linkage(X, 'ward')

plt.figure(figsize=(10, 7))

dendrogram(linked, orientation='top', distance\_sort='descending', show\_leaf\_counts=True)

plt.title("Dendrogram")

plt.show()

