**Data mining using python lab manual**

**1.Demonstrate the following data preprocessing tasks using python libraries.**

**a.->Need of softwares and libraries**

**1.First install java and python softwares in windows and set the path.And check the versions of sw.**

**2.open command prompt type**

**Check the java version:java--version**

**Check the python version: python - –version**

**Py -m pip install numpy**

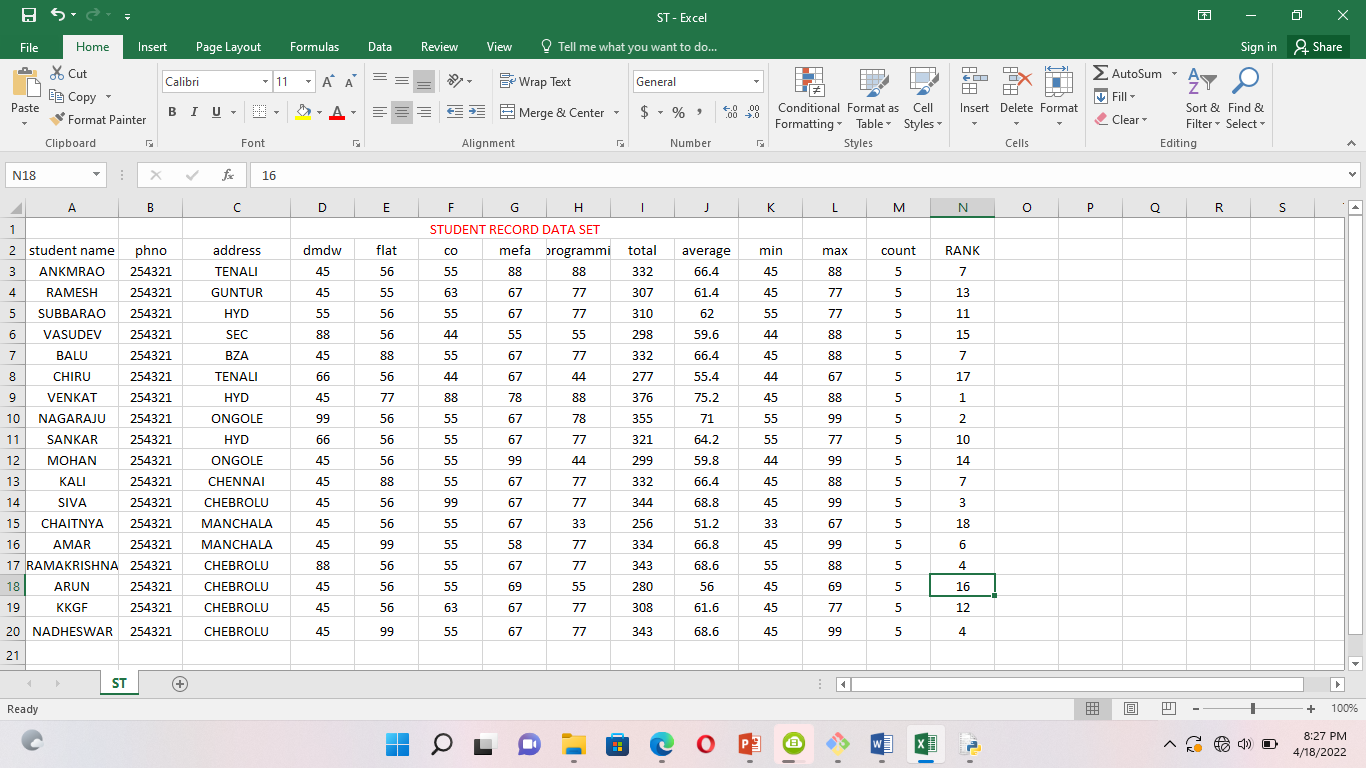
**Py -m pip install –upgrade pip**

**Py -m pip install pandas**

**Py -m pip install matplotlib**

**Above those python libraries are used for datapreprocessing**

**b.Creating the dataset .csv by using Ms-Excel.**



**c.AIM:. LOADING THE DATASET.csv file ?**

**Program: file is saved in data.py**

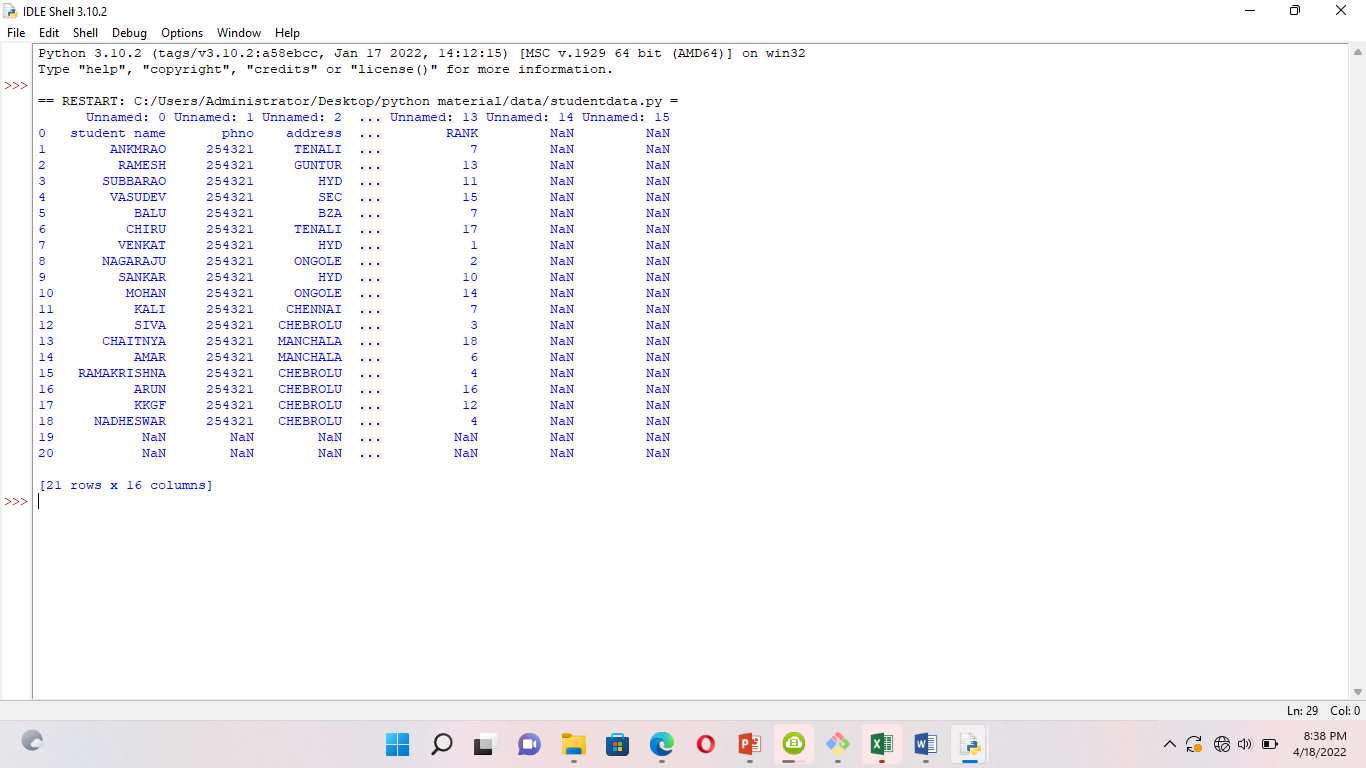
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read\_csv("C:/Users/Administrator/Desktop/python material/data/student record.csv")

print(df.head())

output:

**2.a.AIM.identifying the dependent and independent variables.?**

**Program:**

**import pandas as pd**

**importnumpy as np**

**from sklearn.impute import SimpleImputer # used for handling missing data**

**from sklearn.preprocessing import LabelEncoder, OneHotEncoder # used for encoding categorical data**

**fromsklearn.model\_selection import train\_test\_split # used for splitting training and testing data**

**fromsklearn.preprocessing import StandardScaler # used for feature scaling**

**dataset = pd.read\_csv("C:/Users/Administrator/Desktop/python material/data/Dataset.csv")**

**# to import the dataset into a variable**

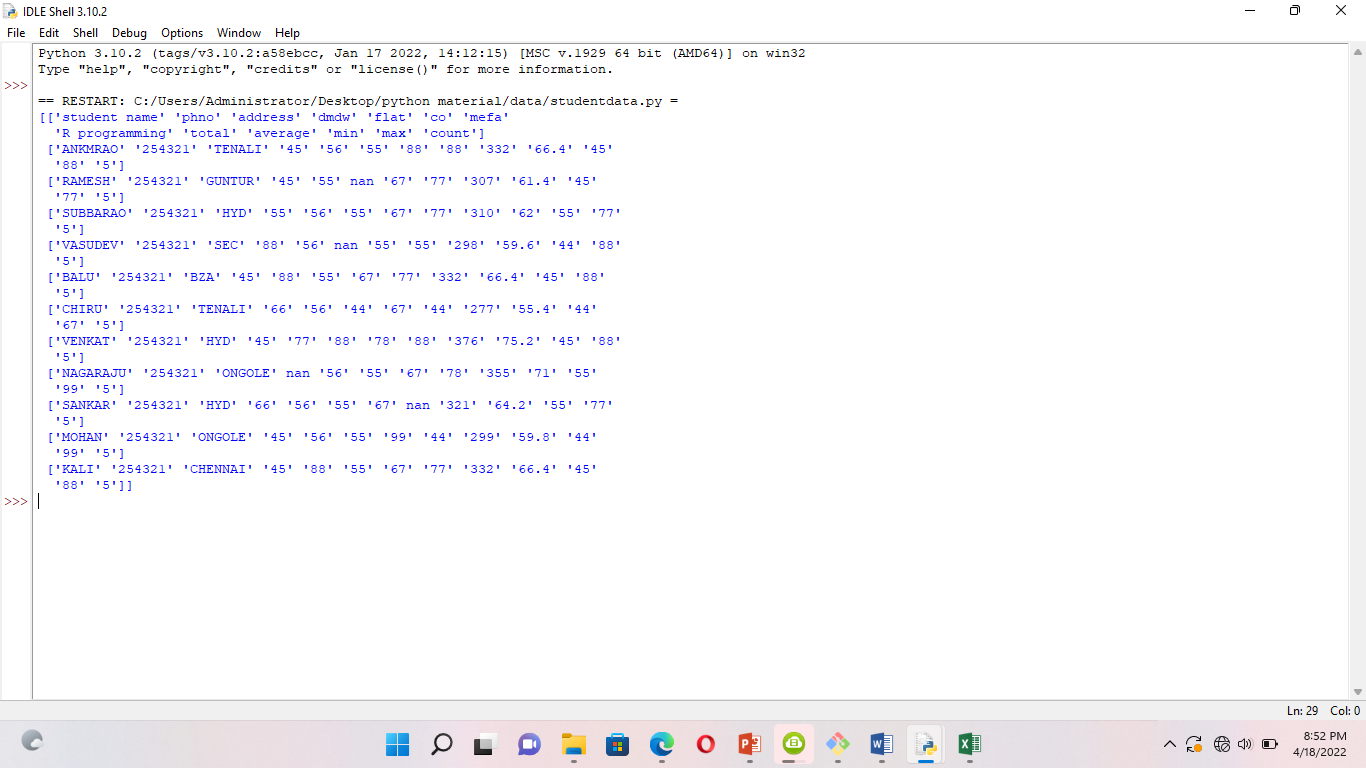
**# Splitting the attributes into independent and dependent attributes**

**X = dataset.iloc[:, :-1].values # attributes to determine dependent variable / Class**

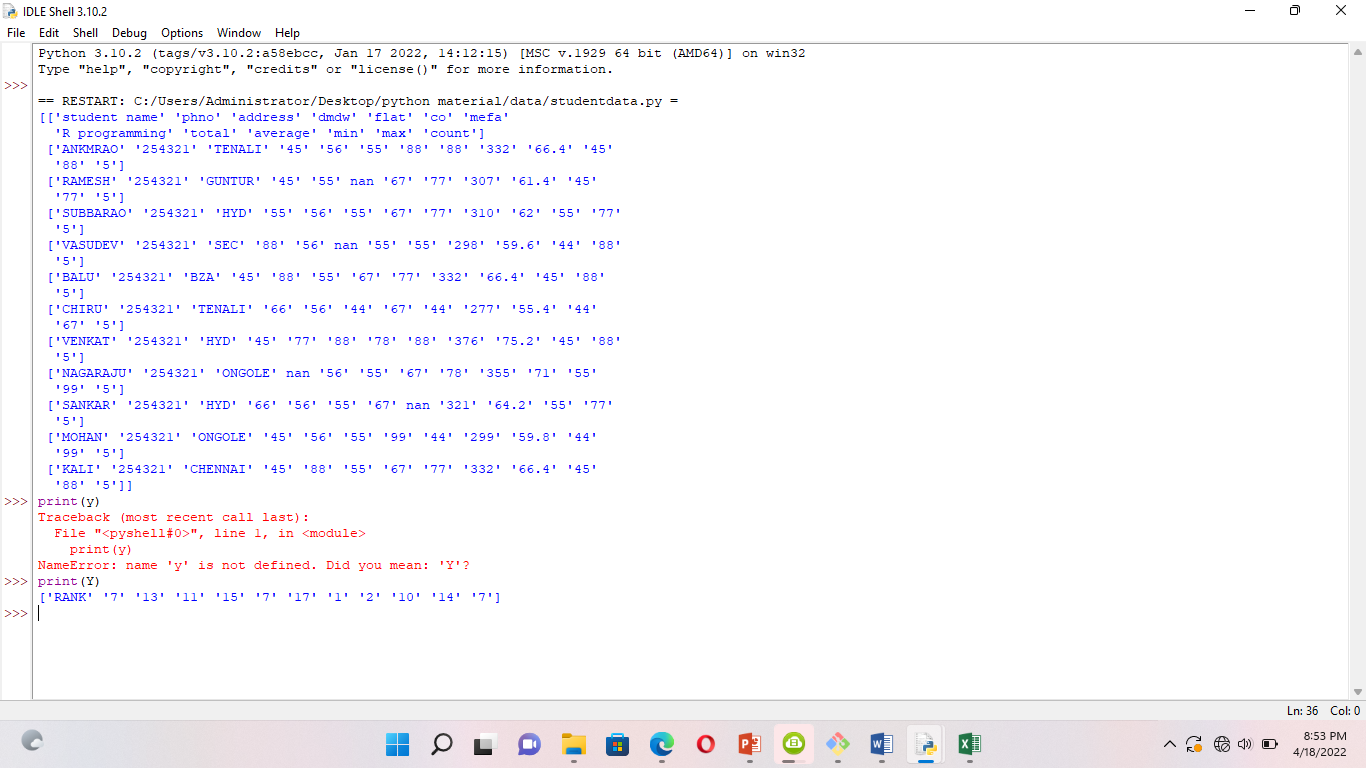
**Y = dataset.iloc[:, -1].values # dependent variable / Class**

**print(dataset)**

**print(X)**



**print(Y)**



# Single selections using iloc and DataFrame

# Rows:

data.iloc[0] # first row of data frame (AleshiaTomkiewicz) - Note a Series data type output.

data.iloc[1] # second row of data frame (Evan Zigomalas)

data.iloc[-1] # last row of data frame (MiRichan)

# Columns:

data.iloc[:,0] # first column of data frame (first\_name)

data.iloc[:,1] # second column of data frame (last\_name)

data.iloc[:,-1] # last column of data frame (id)

# Multiple row and column selections using iloc and DataFrame

data.iloc[0:5] # first five rows of dataframe

data.iloc[:, 0:2] # first two columns of data frame with all rows

data.iloc[[0,3,6,24], [0,5,6]] # 1st, 4th, 7th, 25th row + 1st 6th 7th columns.

data.iloc[0:5, 5:8] # first 5 rows and 5th, 6th, 7th columns of data frame (county -> phone1).

**#To remove the column**

**Col\_to\_drop=[“phno”,”address”]**

**Dataset=dataset.drop(col\_to\_drop,axis=1)**

**Print(dataset)**

**b.data visualization using python**

**Need library is seaborn it is install in windows by using cmd**

**Py -m pip install seaborn**

**Program:**

**import numpy as np**

**import pandas as pd**

**importmatplotlib.pyplot as plt**

**importseaborn as sns**

**df = sns.load\_dataset('iris')**

**#without regression**

**sns.pairplot(df, kind="scatter")**

**plt.show()**

**output:**

**3.AIM: Similarity Disimilarity Measures implementation in Python**

## A. Euclidean Distance Implementation in python

#!/usr/bin/env python

from math import\*

def euclidean\_distance(x,y):

return sqrt(sum(pow(a-b,2) for a, b in zip(x, y)))

print (euclidean\_distance([0,3,4,5],[7,6,3,-1]))

## B.Manhattan distance implementation in Python

#!/usr/bin/env python

from math import\*

def manhatta\_distance(x,y):

return sum(abs(a-b) for a,b in zip(x,y))

print (manhattan\_distance([10,20,10],[10,20,20]))

## C.Minkowsi distance implementation in python

#!/usr/bin/env python

from math import\*

from decimal import Decimal

def nth\_root(value, n\_root):

root\_value = 1/float(n\_root)

return round (Decimal(value) \*\* Decimal(root\_value),3)

def minkowski\_distance(x,y,p\_value):

return nth\_root(sum(pow(abs(a-b),p\_value) for a,b in zip(x, y)),p\_value)

print manhattan\_distance([10,20,10],[10,20,20])

## D. COSINE Similarity implementation in python

#!/usr/bin/env python

from math import\*

def square\_rooted(x):

return round(sqrt(sum([a\*a for a in x])),3)

def cosine\_similarity(x,y):

numerator = sum(a\*b for a,b in zip(x,y))

denominator = square\_rooted(x)\*square\_rooted(y)

return round(numerator/float(denominator),3)

print cosine\_similarity([3, 45, 7, 2], [2, 54, 13, 15])

## E.Jaccard Similarity implementation in python

#!/usr/bin/env python

from math import\*

def jaccard\_similarity(x,y):

intersection\_cardinality = len(set.intersection(\*[set(x), set(y)]))

union\_cardinality = len(set.union(\*[set(x), set(y)]))

return intersection\_cardinality/float(union\_cardinality)

print (jaccard\_similarity([0,1,2,5,6],[0,2,3,5,7,9]))

## 

**4.AIM: write a program to step by step procedure of DECISION TREE visualize Using python.**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**#% matplotlib inline**

**from sklearn.datasets import load\_iris**

**from sklearn import tree**

**clf=tree.DecisionTreeClassifier(random\_state=0)**

**iris=load\_iris()**

**clf=clf.fit(iris.data,iris.target)**

**plt.figure(figsize=(15,10))**

**tree.plot\_tree(clf,filled=True)**

**plt.show()**

## 

## 5.Aim: write a program on linear regression using python.

## Procedure:

## import numpy as np

## import matplotlib.pyplot as plt

## import pandas as pd

## from sklearn.linear\_model import LinearRegression

## X=np.array([1,10,20,40,50,70,80,90,120])

## Y=np.array([3,20,90,110,130,170,150,200,260])

## linreg=LinearRegression()

## x=X.reshape(-1,1)

## linreg.fit(x,Y)

## Y\_pred=linreg.predict(x)

## plt.scatter(x,Y)

## plt.plot(x,Y\_pred,color="red")

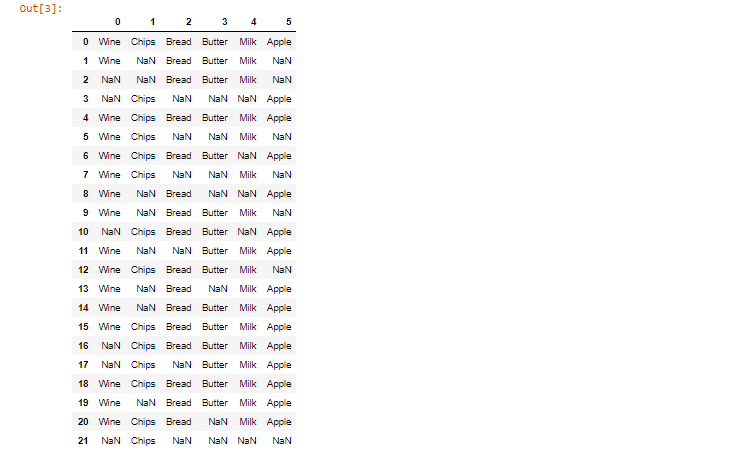
## plt.show()

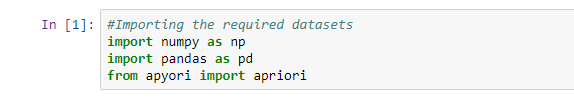
## 

## 6.Aim: write program easy way step by step procedure of apriori algorithm using python.

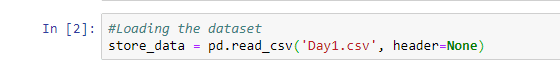
## Procedure:

#### **Step 1: Import the libraries**

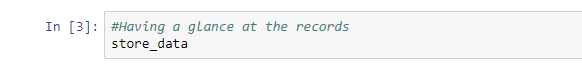




#### **Step 2: Load the dataset**



#### **Step 3: Have a glance at the records**

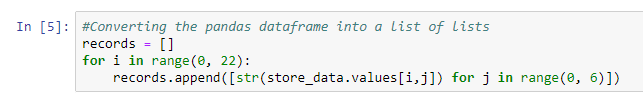


#### **Step 4: Look at the shape**

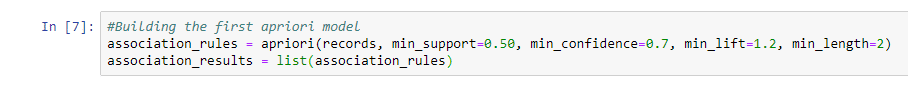
https://intellipaat.com/blog/wp-content/uploads/2019/05/Apiori12.png

https://intellipaat.com/blog/wp-content/uploads/2019/05/Apiori13.png

#### **Step 5:** **Convert Pandas DataFrame into a list of lists**



#### **Step 6: Build the Apriori model**



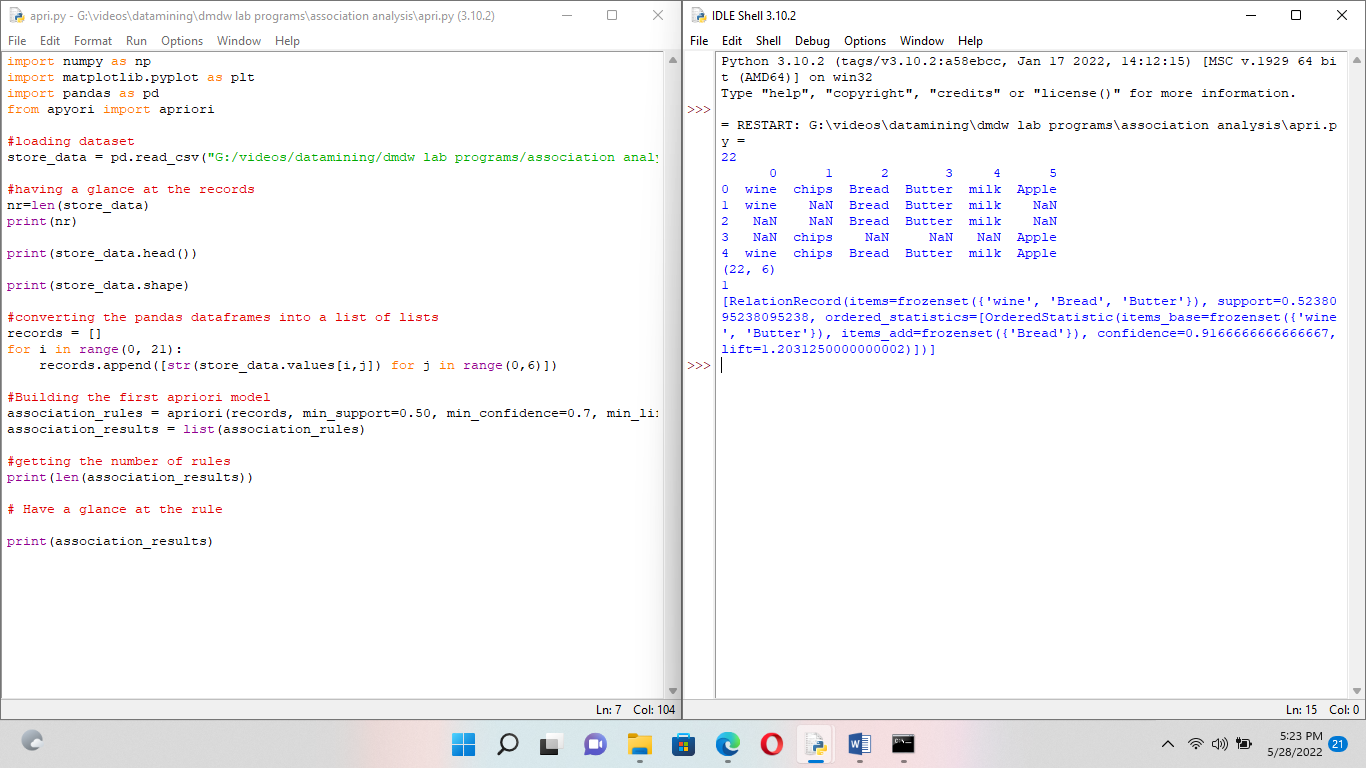
#### **Step 7: Print out the number of rules**

### **https://intellipaat.com/blog/wp-content/uploads/2019/05/Apiori16.png**

#### **Step 8: Have a glance at the rule**



https://intellipaat.com/blog/wp-content/uploads/2019/05/Apiori18.png



**7.Aim: write a program for step by step procedure of k-means algorithm using python**

**Procedure:**

**#import required libraries**

**import numpy as np**

**import matplotlib.pyplot as plot**

**from sklearn.datasets import make\_blobs**

**#Generate our dataset**

**dataset=make\_blobs(n\_samples=200,centers=4,n\_features=2,cluster\_std=1.6.random\_state=50)**

**points=dataset[0]**

**#import kmeans**

**fromsklearn.cluster import KMeans**

**#create a kmeans objects**

**kmeans=KMeans(n\_cluster=4)**

**#fit the Kmeans objects to the dataset**

**kmeans.fit(points)**

**plt.scatter(dataset[0][:,0],dataset[0][:,1])**

**#create cluster**

**clusters=kmeans.cluster\_centers\_**

**#print out the clusters**

**print(clusters)**

**y\_km=kmeans.fit\_predict(points)**

**points**

**plt.scatter(points[y\_km==0,0],points[y\_km==0,1],s=50,color='red')**

**plt.scatter(points[y\_km==1,0],points[y\_km==1,1],s=50,color='green')**

**plt.scatter(points[y\_km==2,0],points[y\_km==2,1],s=50,color='yellow')**

**plt.scatter(points[y\_km==3,0],points[y\_km==3,1],s=50,color='cyan')**

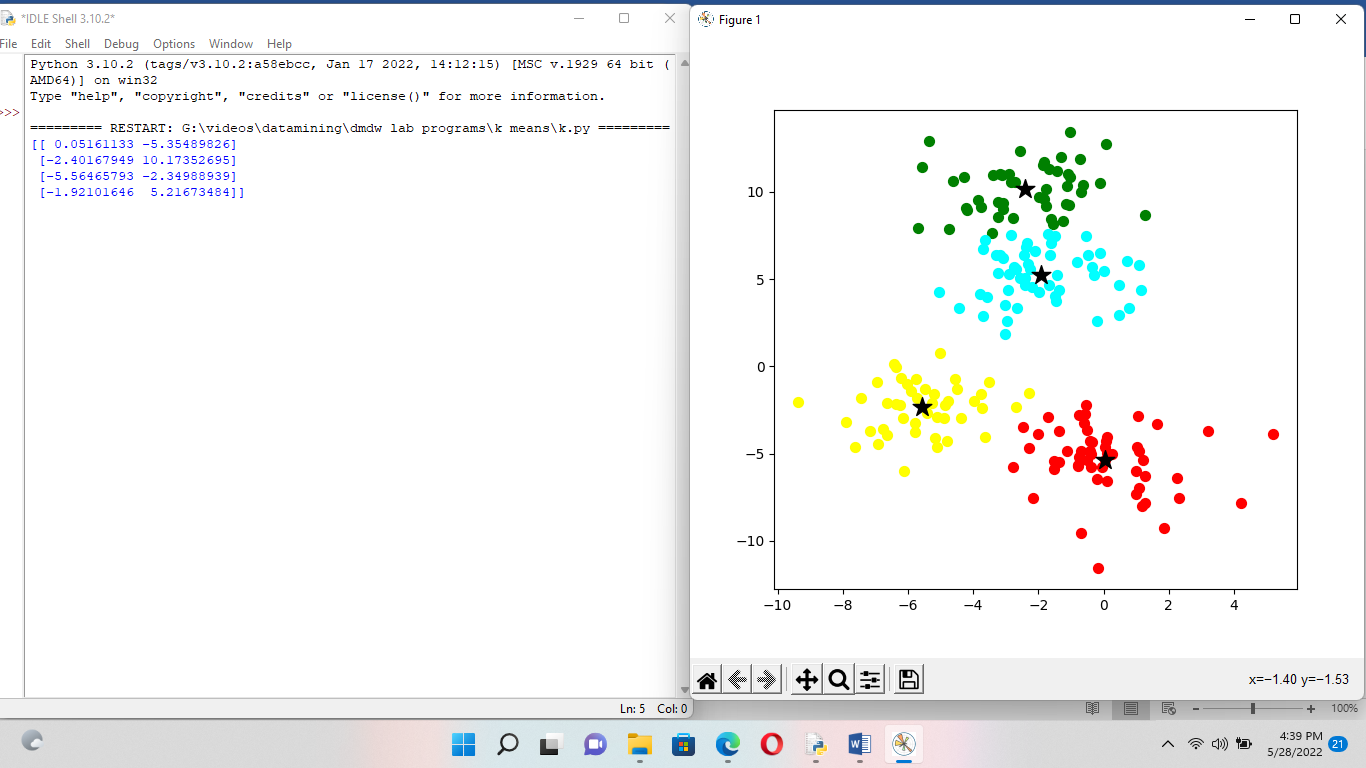
**plt.scatter(clusters[0][0],clusters[0][1],marker='\*',s=200,color='black')**

**plt.scatter(clusters[1][0],clusters[1][1],marker='\*',s=200,color='black')**

**plt.scatter(clusters[2][0],clusters[2][1],marker='\*',s=200,color='black')**

**plt.scatter(clusters[3][0],clusters[3][1],marker='\*',s=200,color='black')**

**plt.show()**



**8.Aim: write a program for step by step procedure of hierarchical clustering algorithm using python.?**

**# importing the libraries**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**# Importing the dataset**

**dataset = pd.read\_csv("G:/videos/datamining/dmdw lab programs/Wholesale customers data.csv")**

**X = dataset.iloc[:, [3, 4]].values**

**from sklearn.preprocessing import normalize**

**data\_scaled = normalize(dataset)**

**data\_scaled = pd.DataFrame(data\_scaled, columns=dataset.columns)**

**data\_scaled.head()**

**X=data\_scaled**

**X.head()**

**# Using the dendrogram to find the optimal number of clusters**

**import scipy.cluster.hierarchy as sch**

**dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))**

**plt.title('Dendrogram')**

**plt.xlabel('Customers')**

**plt.ylabel('Euclidean distances')**

**plt.show()**

**# Training the Hierarchical Clustering model on the dataset**

**from sklearn.cluster import AgglomerativeClustering**

**hc = AgglomerativeClustering(n\_clusters = 2, affinity = 'euclidean', linkage = 'ward')**

**y\_hc = hc.fit\_predict(X)**

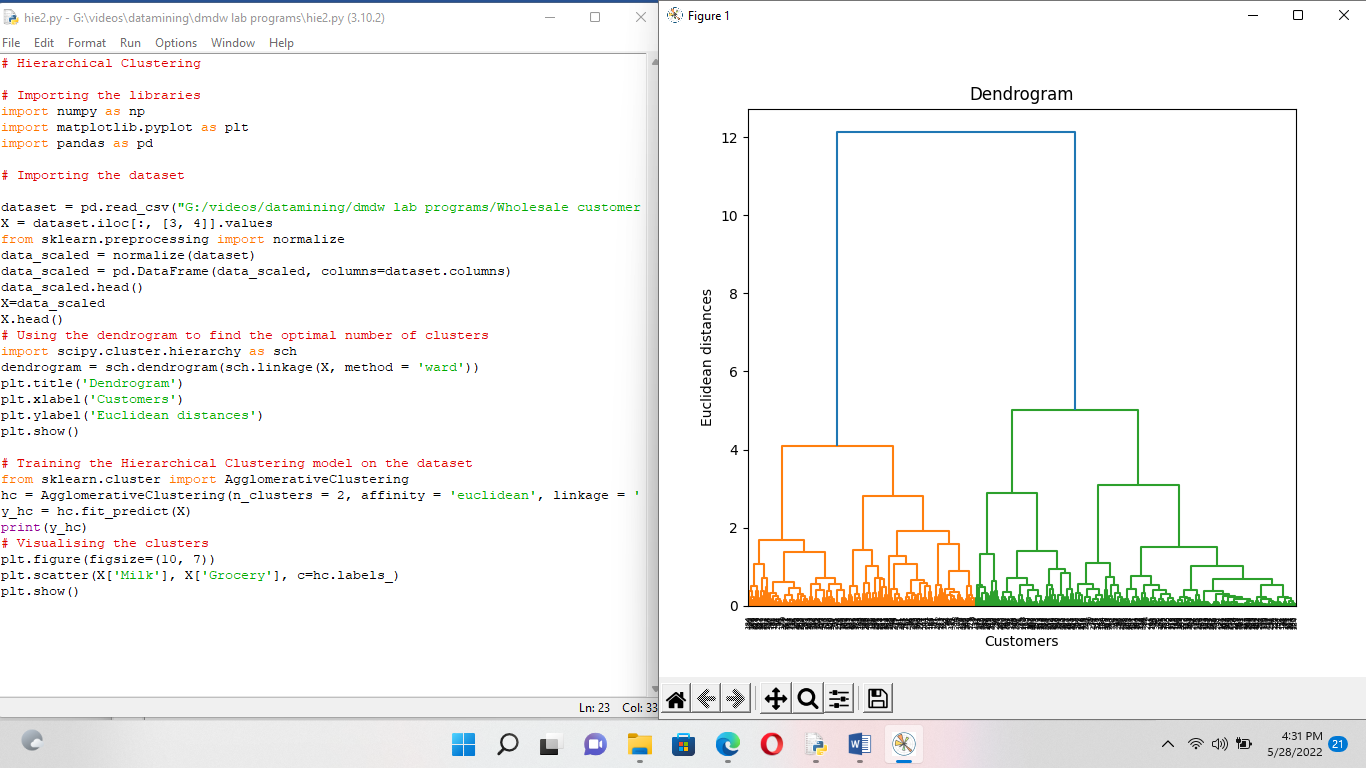
**print(y\_hc)**

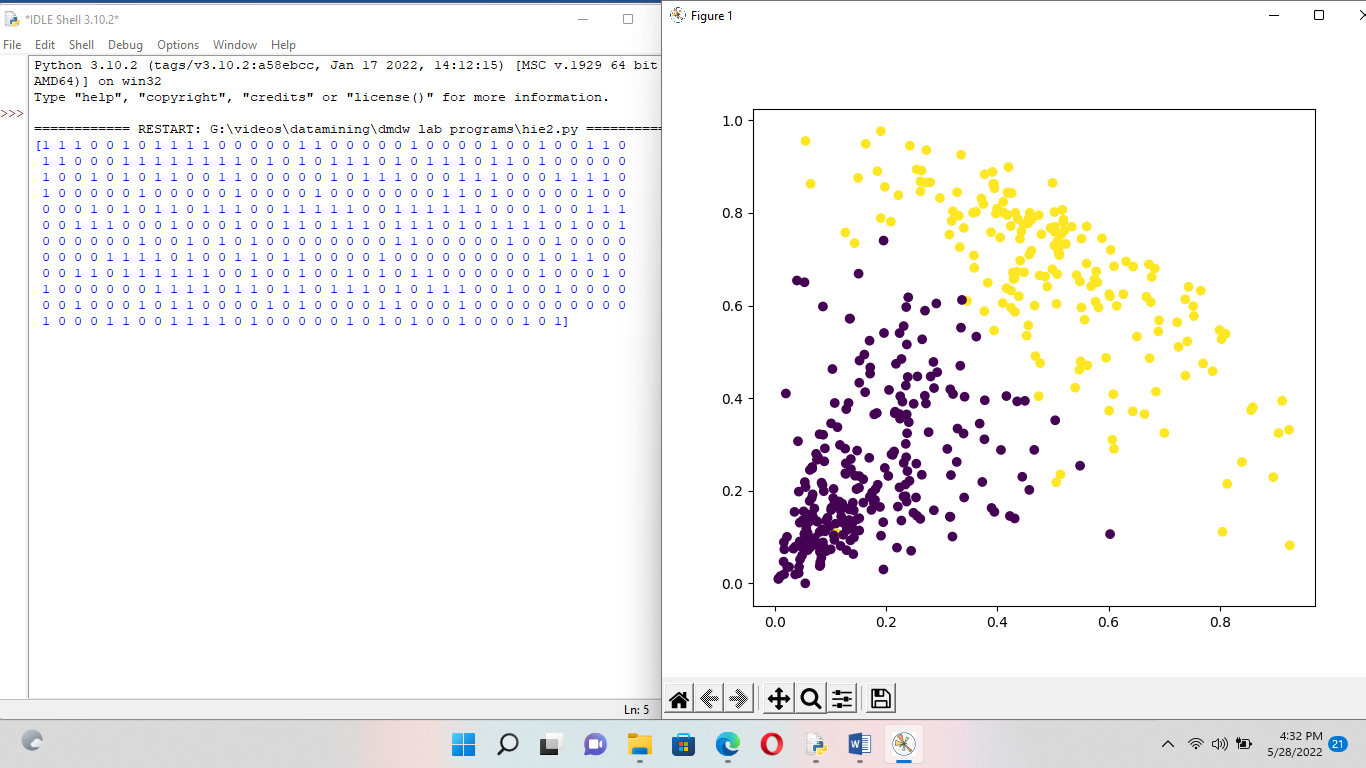
**# Visualising the clusters**

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

**plt.scatter(X['Milk'], X['Grocery'], c=hc.labels\_)**

**plt.show()**





**9.Aim: write program for a step by step procedure of DBScan algorithm using python.?**

**Procedure:**

**import numpy as np**

**from sklearn.datasets import make\_blobs**

**from matplotlib import pyplot as plt**

**from pandas import DataFrame**

**X, \_ = make\_blobs(n\_samples = 500, centers = 3, n\_features = 2, random\_state = 20)**

**df = DataFrame(dict(x=X[:,0], y=X[:,1]))**

**fig, ax = plt.subplots(figsize=(8,8))**

**df.plot(ax=ax, kind='scatter', x='x', y='y')**

**plt.xlabel('x\_1')**

**plt.ylabel('x\_2')**

**plt.show()**

**#We are just getting the labels of the clusters in the next step.**

**from sklearn.cluster import DBSCAN**

**clustering = DBSCAN(eps = 1, min\_samples = 5).fit(X)**

**cluster = clustering.labels\_**

**#To see how many clusters has it found on the dataset**

**len(set(cluster))**

**# You can see we just visualize our clusters by this simple piece of code .**

**def show\_clusters(X, cluster):**

**df = DataFrame(dict(x=X[:,0], y=X[:,1], label=cluster))**

**colors = {-1: 'red', 0: 'blue', 1:'orange', 2:'green', 3:'yellow'}**

**fig, ax = plt.subplots(figsize=(8,8))**

**grouped = df.groupby('label')**

**for key, group in grouped:**

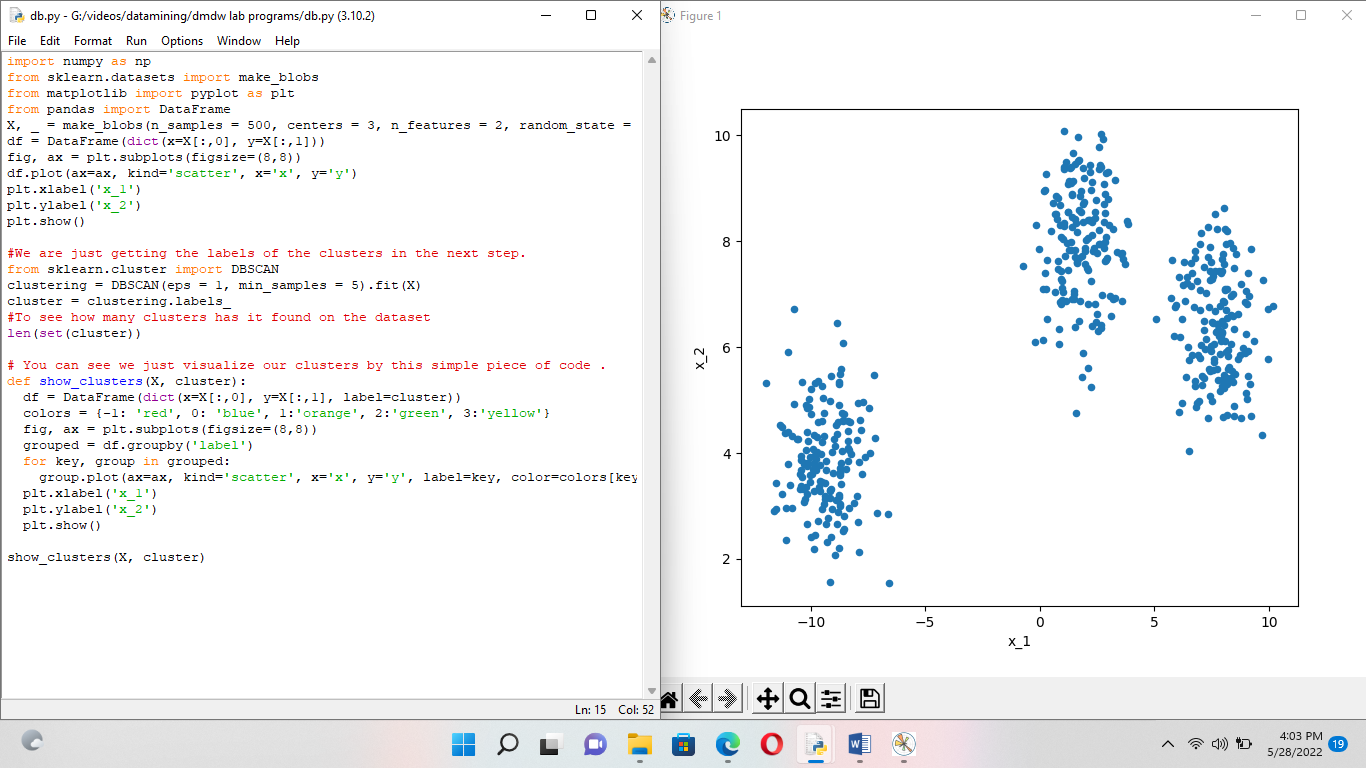
**group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])**

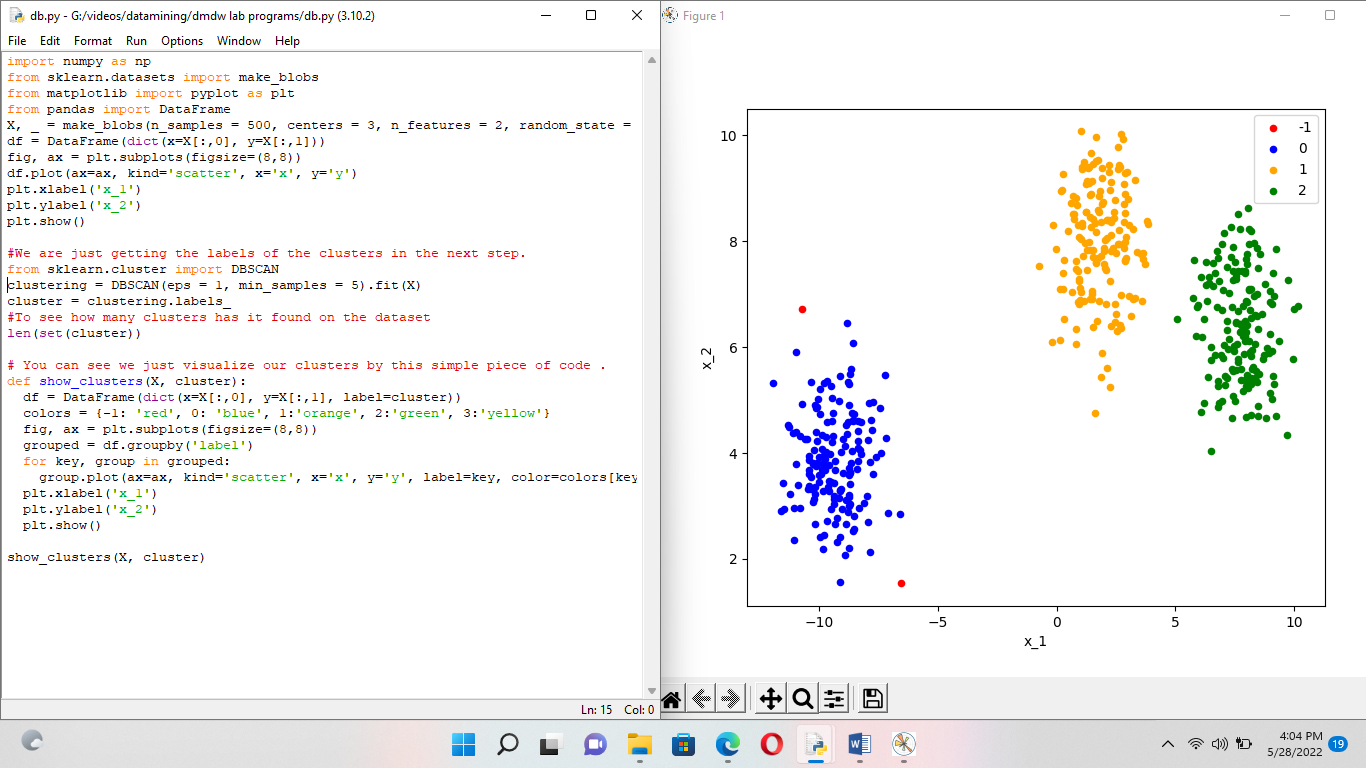
**plt.xlabel('x\_1')**

**plt.ylabel('x\_2')**

**plt.show()**

**show\_clusters(X, cluster)**





**10.Aim:write a program for step by step procedure of apply navie bayes classifier on any dataset using python.?**

**Procedure:**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

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

**from sklearn.naive\_bayes import GaussianNB**

**#% matplotlib inline**

**from sklearn.datasets import load\_iris**

**##data=load\_iris()**

**data=pd.read\_csv("G:/videos/datamining/dmdw lab programs/Iris.csv")**

**print(data)**

**x=data.iloc[:,:-1].values**

**y=data.iloc[:,-1].values**

**print(x,y)**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.4,random\_state=50)**

**print(x\_train,y\_train)**

**(y\_train==0).sum()**

**(y\_train==1).sum()**

**(y\_train==2).sum()**

**gnb=GaussianNB()**

**gnb.fit(x\_train,y\_train)**

**y\_pred=gnb.predict(x\_test)**

**print("total number of instances:(x\_test.shape(0)) Incorrect Guesses:(y\_test!=y\_pred).sum())")**

