LAKIREDDY BALIREDDY COLLEGE OF ENGINEERING

(Autonomus)

Department of Computer Science and Engineering

20CS58 Data Mining using Python lab

Name of Student:

Registered No :

Section: A

Academic Year: 2021-2022

INDEX

AIM:- Demonstrate the following data preprocessing tasks using python libraries.

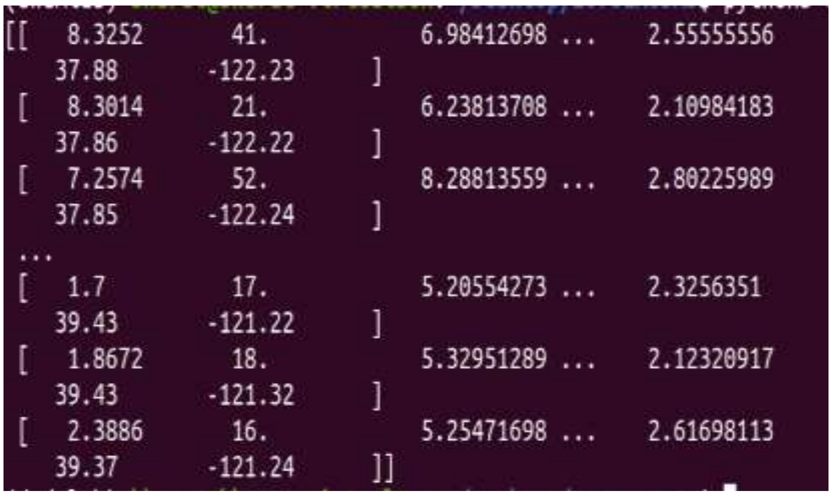
A)Loading the dataset.

DESC:-
Data loading is the process of copying and loading data or data sets from a source file, folder or application to a database or similar application. It is usually implemented by copying digital data from a source and pasting or loading the data to a data storage or processing utility.

PROGRAM:-

```
from sklearn import datasets

from sklearn.datasets import fetch_california_housing

df=fetch_california_housing()

x=df.data

print(x)
```

OUTPUT:-



```
[[   8.3252      41.           6.98412698 ...    2.55555556
    37.88      -122.23      ]
 [   8.3014      21.           6.23813708 ...    2.10984183
    37.86      -122.22      ]
 [   7.2574      52.           8.28813559 ...    2.80225989
    37.85      -122.24      ]
 ...
 [   1.7         17.           5.20554273 ...    2.3256351
    39.43      -121.22      ]
 [   1.8672      18.           5.32951289 ...    2.12320917
    39.43      -121.32      ]
 [   2.3886      16.           5.25471698 ...    2.61698113
    39.37      -121.24      ]]
```

Result:- Program Executed Successfully.

B)Identifying the dependent and independent variables.

DESC:-
Independent variables (also referred to as Features) are the input for a process that is being analyzes. Dependent variables are the output of the process.

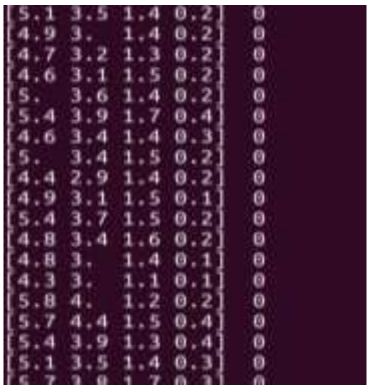
For example, in the below data set, the independent variables are the input of the purchasing process being analyzed. The result (whether a user purchased or not) is the dependent variable

PROGRAM:-

```
from sklearn.datasets import load_iris

i=load_iris()

X,Y=i.data,i.target

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

```
                    print(X[i],"",Y[i])
```

Output:-



```
[5.1 3.5 1.4 0.2]   0
[4.9 3.  1.4 0.2]   0
[4.7 3.2 1.3 0.2]   0
[4.6 3.1 1.5 0.2]   0
[5.  3.6 1.4 0.2]   0
[5.4 3.9 1.7 0.4]   0
[4.6 3.4 1.4 0.3]   0
[5.  3.4 1.5 0.2]   0
[4.4 2.9 1.4 0.2]   0
[4.9 3.1 1.5 0.1]   0
[5.4 3.7 1.5 0.2]   0
[4.8 3.4 1.6 0.2]   0
[4.8 3.  1.4 0.1]   0
[4.3 3.  1.1 0.1]   0
[5.8 4.  1.2 0.2]   0
[5.7 4.4 1.5 0.4]   0
[5.4 3.9 1.3 0.4]   0
[5.1 3.5 1.4 0.3]   0
```

Result:- Program Executed Successfully.

C)Dealing with missing data

DESC:-

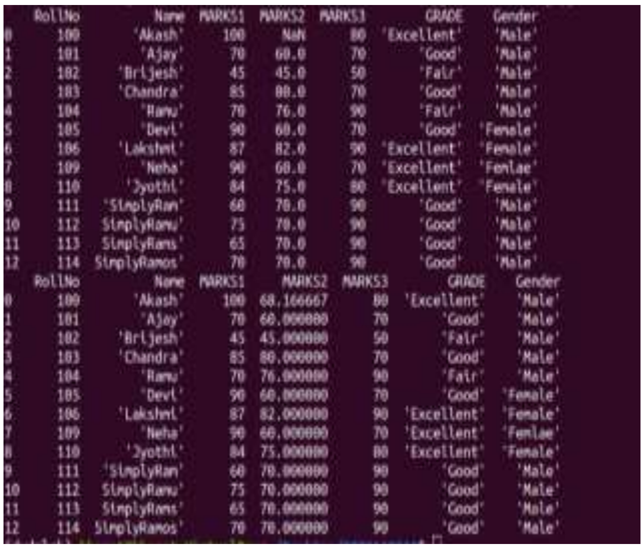
Missing data is defined as the values or data that is not stored (or not present) for some variable/s in the given dataset.

Below is a sample of the missing data from the Titanic dataset. You can see the columns 'Age' and 'Cabin' have some missing values.

PROGRAM:-

```
          import numpy as np

          import pandas as pd

          df=pd.read_csv('stu.csv')

          print(df)

          df['MARKS2']=df['MARKS2'].fillna(df['MARKS2'].mean())

         print(df)
```

OUTPUT:-



```
    RollNo          Name  MARKS1  MARKS2  MARKS3        GRADE    Gender
0      100       'Akash'     100     NaN      80  'Excellent'    'Male'
1      101        'Ajay'      70    60.0      70       'Good'    'Male'
2      102     'Brijesh'      45    45.0      50       'Fair'    'Male'
3      103     'Chandra'      85    80.0      70       'Good'    'Male'
4      104        'Ramu'      70    76.0      90       'Fair'    'Male'
5      105        'Devi'      90    60.0      70       'Good'  'Female'
6      106     'Lakshmi'      87    82.0      90  'Excellent'  'Female'
7      109        'Neha'      90    60.0      70  'Excellent'  'Fenlae'
8      110      'Jyothi'      84    75.0      80  'Excellent'  'Female'
9      111   'SimplyRam'      60    70.0      90       'Good'    'Male'
10     112   SimplyRamu'      75    70.0      90       'Good'    'Male'
11     113   SimplyRams'      65    70.0      90       'Good'    'Male'
12     114  SimplyRamos'      70    70.0      90       'Good'    'Male'
    RollNo          Name  MARKS1     MARKS2  MARKS3        GRADE    Gender
0      100       'Akash'     100  68.166667      80  'Excellent'    'Male'
1      101        'Ajay'      70  60.000000      70       'Good'    'Male'
2      102     'Brijesh'      45  45.000000      50       'Fair'    'Male'
3      103     'Chandra'      85  80.000000      70       'Good'    'Male'
4      104        'Ramu'      70  76.000000      90       'Fair'    'Male'
5      105        'Devi'      90  60.000000      70       'Good'  'Female'
6      106     'Lakshmi'      87  82.000000      90  'Excellent'  'Female'
7      109        'Neha'      90  60.000000      70  'Excellent'  'Fenlae'
8      110      'Jyothi'      84  75.000000      80  'Excellent'  'Female'
9      111   'SimplyRam'      60  70.000000      90       'Good'    'Male'
10     112   SimplyRamu'      75  70.000000      90       'Good'    'Male'
11     113   SimplyRams'      65  70.000000      90       'Good'    'Male'
12     114  SimplyRamos'      70  70.000000      90       'Good'    'Male'
```

Result:- Program Executed Successfully.

AIM:- Demonstrate the following data preprocessing tasks using python libraries.

A)Dealing with categorical data.

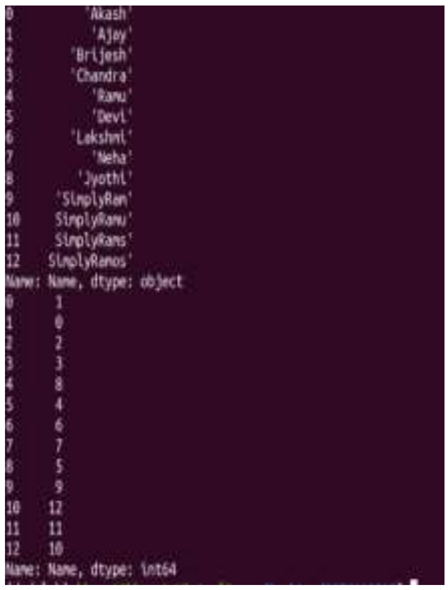
DESC:-

Categorical data is the statistical data comprising categorical variables of data that are converted into categories. One of the examples is a grouped data. More precisely, categorical data could be derived from qualitative data analysis that are countable, or from quantitative data analysis grouped within given intervals. These data are summarised in the form of a probability table. However, when we consider data analysis, it is referred to use the term “categorical data”, which is applied to data sets. Also, it is to be noted that, while containing some categorical variables, the data set may also contain non-categorical variables.

PROGRAM:-

```
#using  LabelEncoder
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
df=pd.read_csv('stu.csv')
print(df['Name'])
a=LabelEncoder()
df['Name']=a.fit_transform(df['Name'])
print(df['Name'])
```

OUTPUT:-



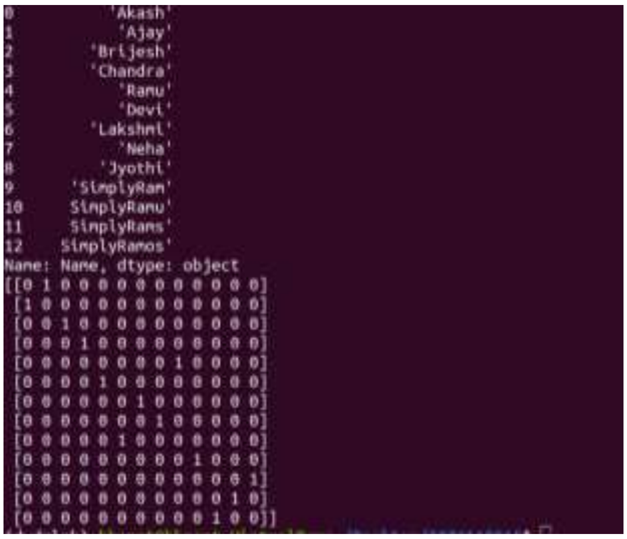
```
0          'Akash'
1           'Ajay'
2        'Brijesh'
3        'Chandra'
4           'Ramu'
5           'Devi'
6        'Lakshmi'
7           'Neha'
8         'Jyothi'
9      'SimplyRam'
10     SimplyRamu'
11     SimplyRams'
12    SimplyRamos'
Name: Name, dtype: object
0     1
1     0
2     2
3     3
4     8
5     4
6     6
7     7
8     5
9     9
10   12
11   11
12   10
Name: Name, dtype: int64
```

PROGRAM:-

```
#Using LabelBinarizer
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer
df=pd.read_csv('stu.csv')
print(df['Name'])
a=LabelBinarizer()
df=a.fit_transform(df['Name'])
```

```
print(df)
```

OUTPUT:-

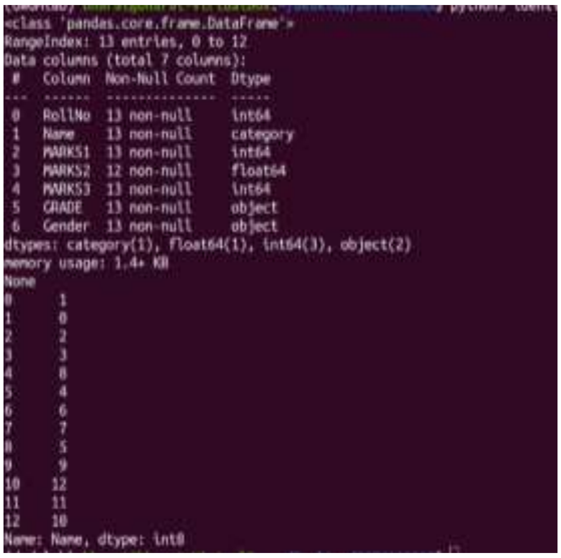


```
0          'Akash'
1           'Ajay'
2        'Brijesh'
3        'Chandra'
4           'Ramu'
5           'Devi'
6        'Lakshmi'
7           'Neha'
8         'Jyothi'
9      'SimplyRam'
10     SimplyRamu'
11     SimplyRams'
12    SimplyRamos'
Name: Name, dtype: object
[[0 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 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 1 0 0 0 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 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 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 1 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0]]
```

PROGRAM:-

```
#Using asType,cat.Codes

import numpy as np

import pandas as pd

df=pd.read_csv('stu.csv')

df['Name']=df['Name'].astype('category')

print(df.info())

df['Name']=df['Name'].cat.codes

print(df['Name'])
```

OUTPUT:-



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13 entries, 0 to 12
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   RollNo  13 non-null     int64
 1   Name    13 non-null     category
 2   MARKS1  13 non-null     int64
 3   MARKS2  12 non-null     float64
 4   MARKS3  13 non-null     int64
 5   GRADE   13 non-null     object
 6   Gender  13 non-null     object
dtypes: category(1), float64(1), int64(3), object(2)
memory usage: 1.4+ KB
None
0      1
1      0
2      2
3      3
4      8
5      4
6      6
7      7
8      5
9      9
10    12
11    11
12    10
Name: Name, dtype: int8
```

Result:- Program Executed Successfully.

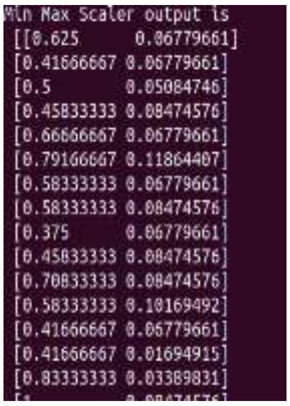
B)Scaling the features

PROGRAM:-

```
import pandas as pd
```

```
from sklearn.preprocessing import MinMaxScaler,StandardScaler
df = pd.read_csv('iris.csv')
x = df.iloc[:, 1:3].values
min_max = MinMaxScaler(feature_range =(0, 1))
min_max1=StandardScaler()
x_after_min_max = min_max.fit_transform(x)
x_after_min_max1=min_max1.fit_transform(x)
print("Min Max Scaler output is\n", x_after_min_max)
print("Standard Scaler output is\n",x_after_min_max1)
```

OUTPUT:-



```
Min Max Scaler output is
 [[0.625      0.06779661]
 [0.41666667 0.06779661]
 [0.5        0.05084746]
 [0.45833333 0.08474576]
 [0.66666667 0.06779661]
 [0.79166667 0.11864407]
 [0.58333333 0.06779661]
 [0.58333333 0.08474576]
 [0.375      0.06779661]
 [0.45833333 0.08474576]
 [0.70833333 0.08474576]
 [0.58333333 0.10169492]
 [0.41666667 0.06779661]
 [0.41666667 0.01694915]
 [0.83333333 0.03389831]
```

Result:- Program Executed Successfully.

C)Splitting dataset into Training and Testing Sets

DESC:-

Training Data

The observations in the training set form the experience that the algorithm uses to learn. In supervised learning problems, each observation consists of an observed output variable and one or more observed input variables.

Test Data

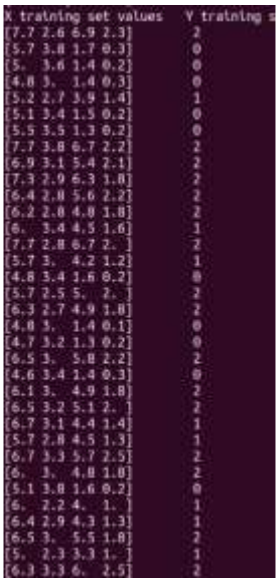
The test set is a set of observations used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

PROGRAM:-

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
i=load_iris()
X,Y=i.data,i.target
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=1)
print("X training set values\tY training set values")
for i in range(0,len(X_train)):
```

```
            print(X_train[i],'\t',Y_train[i])
```

OUTPUT:-



```
X training set values   Y training s
[7.7 2.6 6.9 2.3]         2
[5.7 3.8 1.7 0.3]         0
[5.  3.6 1.4 0.2]         0
[4.8 3.  1.4 0.3]         0
[5.2 2.7 3.9 1.4]         1
[5.1 3.4 1.5 0.2]         0
[5.5 3.5 1.3 0.2]         0
[7.7 3.8 6.7 2.2]         2
[6.9 3.1 5.4 2.1]         2
[7.3 2.9 6.3 1.8]         2
[6.4 2.8 5.6 2.2]         2
[6.2 2.8 4.8 1.8]         2
[6.  3.4 4.5 1.6]         1
[7.7 2.8 6.7 2. ]         2
[5.7 3.  4.2 1.2]         1
[4.8 3.4 1.6 0.2]         0
[5.7 2.5 5.  2. ]         2
[6.3 2.7 4.9 1.8]         2
[4.8 3.  1.4 0.1]         0
[4.7 3.2 1.3 0.2]         0
[6.5 3.  5.8 2.2]         2
[4.6 3.4 1.4 0.3]         0
[6.1 3.  4.9 1.8]         2
[6.5 3.2 5.1 2. ]         2
[6.7 3.1 4.4 1.4]         1
[5.7 2.8 4.5 1.3]         1
[6.7 3.3 5.7 2.5]         2
[6.  3.  4.8 1.8]         2
[5.1 3.8 1.6 0.2]         0
[6.  2.2 4.  1. ]         1
[6.4 2.9 4.3 1.3]         1
[6.5 3.  5.5 1.8]         2
[5.  2.3 3.3 1. ]         1
[6.3 3.3 6.  2.5]         2
```

Result:- Program Executed Successfully.

AIM:-Demonstrate the following Similarity and Dissimilarity Measures using python

A)Pearson’s Correlation

DESC:-

The Pearson's Correlation Coefficient is also known as the Pearson Product-Moment Correlation Coefficient. It is a measure of the linear relationship between two random variables - **X** and **Y**. Mathematically, if (**σXY**) is the covariance between **X** and **Y**, and (**σX**) is the standard deviation of **X**, then the Pearson's correlation coefficient **ρ** is given by:

$$\rho X,Y=\sigma XY/\sigma X\sigma Y$$

PROGRAM:-

```
from scipy.stats import pearsonr

X=[-2,-1,0,1,2]

Y=[4,1,3,2,0]

corr=pearsonr(X,Y)

print("pearson correlation:",corr)
```

B)Cosine Similarity

DESC:-

Cosine distance measure for clustering determines the **cosine** of the angle between two vectors given by the following formula.

$$\text{sim}(A,B)=\cos(\theta)=\frac{A\cdot B}{\|A\|B\|}$$

Here (**theta**) gives the angle between two vectors and A, B are n-dimensional vectors.

PROGRAM:-

```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
A="deep learning can be hard"
B="deep learning can be soft"
documents=[A,B]
ob=CountVectorizer()
X=ob.fit_transform(documents)
Y=X.todense()
df=pd.DataFrame(Y,columns=ob.get_feature_names_out(),index=['A','B'])
print(df)
print("similarity matrix:\n",cosine_similarity(df,df))
```

C)Jaccard Similarity

DESC:-

The Jaccard distance measures the similarity of the two data set items as the **intersection** of those items divided by the **union** of the data items.

$$J(A,B) = \frac{|A\cap B|}{|A\cup B|} = \frac{|A\cap B|}{|A|+|B|-|A\cap B|}$$

PROGRAM:-

```
import numpy as np
from scipy.spatial.distance import jaccard
a=np.array([1,0,1,0,0,1])
b=np.array([0,1,0,1,0,1])
print(" jaccard distance:",jaccard(a,b))
```

D)Euclidean Distance

DESC:-

Euclidean distance is considered the traditional metric for problems with geometry. It can be simply explained as the **ordinary distance** between two points. It is one of the most used algorithms in the cluster analysis. One of the algorithms that use this formula would be **K-mean**. Mathematically it computes the **root of squared differences** between the coordinates between two objects.

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$

PROGRAM:-

```
import numpy as np
import pandas as pd

from sklearn.metrics.pairwise import manhattan_distances

X=np.ones((1,2))

Y=np.full((2,2),2)

manhattan_distances(X,Y,sum_over_features=False)
```

E)Manhattan Distance

DESC:-

This determines the absolute difference among the pair of the coordinates.

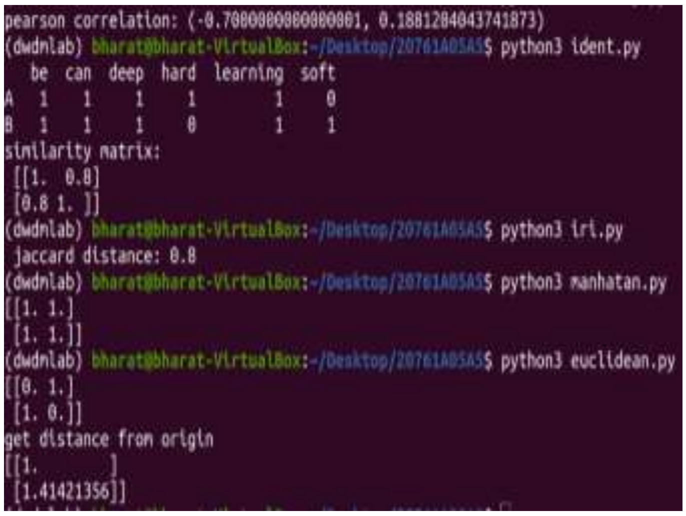
Suppose we have two points P and Q to determine the distance between these points we simply have to calculate the perpendicular distance of the points from X-Axis and Y-Axis.
In a plane with P at coordinate (x1, y1) and Q at (x2, y2).

Manhattan distance between P and Q = |x1 – x2| + |y1 – y2|

PROGRAM:-

```
from sklearn.metrics.pairwise import euclidean_distances

X=[[0,1],[1,1]]

print(euclidean_distances(X,X))

print("get distance from origin")

print(euclidean_distances(X,[[0,0]]))
```

OUTPUT:-



```
pearson correlation: (-0.7000000000000001, 0.1881284043741873)
(dwdnlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 ident.py
   be  can  deep  hard  learning  soft
A   1    1     1     1         1     0
B   1    1     1     0         1     1
similarity matrix:
 [[1.  0.8]
 [0.8 1. ]]
(dwdnlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 iri.py
 jaccard distance: 0.8
(dwdnlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 manhatan.py
[[1. 1.]
 [1. 1.]]
(dwdnlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 euclidean.py
[[0. 1.]
 [1. 0.]]
get distance from origin
[[1.        ]
 [1.41421356]]
```

Result:- Program Executed Successfully.

AIM:-Build a model using linear regression algorithm on any dataset.

DESC:-

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables gettingused.

Hypothesis function for Linear Regression :

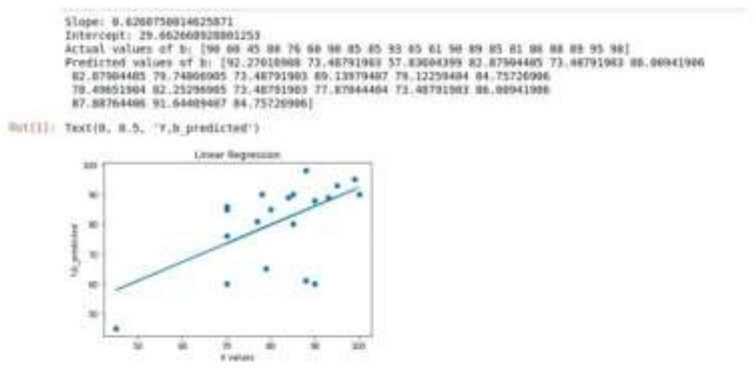
$$y = \theta_1 + \theta_2 . x$$

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2 \qquad J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

PROGRAM:-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
x=np.array([5,10,15,20,25,30]).reshape((-1,1))
y=np.array([5,20,14,32,22,38])
print(x,y)
model=LinearRegression()
model.fit(x,y)
result=model.score(x,y)
print("Score:",result)
print("Intercret:",model.intercept_)
print("slope:",model.coef_)
y_pred=model.predict(x)
print('Actual values of y:',y)
print('predicted values of y:',y_pred)
plt.scatter(x,y,color='black')
plt.plot(x,y_pred,color='blue',linewidth=2,marker='o',markerfacecolor='green')
```

OUTPUT:-



```
Slope: 0.6260750014625871
Intercept: 29.662660928801253
Actual values of b: [90 60 45 88 76 60 98 85 85 93 65 61 90 89 85 81 88 88 89 95 98]
Predicted values of b: [92.27010908 73.48791983 57.83604399 82.87904405 73.48791983 88.00941986
 82.87904405 79.74806905 73.48791983 89.13979487 78.12259404 84.75726906
 78.49651904 82.25296905 73.48791983 77.87944404 73.48791983 88.00941986
 87.88764406 91.64409487 84.75726906]
```

```
Out[1]: Text(0, 0.5, 'Y,b_predicted')
```

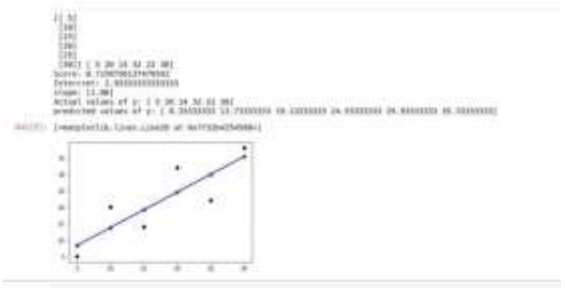
(without using builtin functions)

PROGRAM:-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
n=pd.read_csv('datastudent.csv')
a=np.array(n['MARKS1'])
b=np.array(n['MARKS2'])
n=np.size(a)
```

```
m_a=np.mean(a)
m_b=np.mean(b)
SS_xx=np.sum((m_a-a)*(m_a-a))
SS_xy=np.sum((m_a-a)*(m_b-b))
slope=SS_xy/SS_xx
print("Slope:",slope)
intercept=m_b-(slope*m_a)print("Intercept:",intercept)
b_pred=slope*a+intercept
print("Actual values of b:",b)
print("Predicted values of b:",b_pred)
plt.scatter(a,b)
plt.plot(a,b_pred)
plt.title("Linear Regression")
plt.xlabel("X values")
plt.ylabel("Y,b_predicted")
```

OUTPUT:-



```
[[ 5]
 [10]
 [15]
 [20]
 [25]
 [30]] [ 5 20 14 32 22 38]
Score: [illegible]
Intercept: [illegible]
slope: [illegible]
Actual values of y: [ 5 20 14 32 22 38]
predicted values of y: [illegible]
Out[ ]: [<matplotlib.lines.Line2D at [illegible]>]
```

Result:- Program Executed Successfully.

AIM:-Build a classification model using Decision Tree algorithm on iris dataset

DESC:-

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

1. **Classification trees** (Yes/No types)

What we've seen above is an example of classification tree, where the outcome was a variable like 'fit' or 'unfit'. Here the decision variable is **Categorical**.

1. **Regression trees** (Continuous data types)

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123. **Working** Now that we know what a Decision Tree is, we'll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**. Before discussing the ID3 algorithm, we'll go through few definitions.

- **Entropy:**

Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

of uncertainty or randomness in data. Intuitively, it tells us about the predictability of a certain event. Example, consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there's no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it'll always be heads. In other words, this event has **no randomness** hence it's entropy is zero. In particular, lower values imply less uncertainty while higher values imply high uncertainty.

- **Information Gain:**

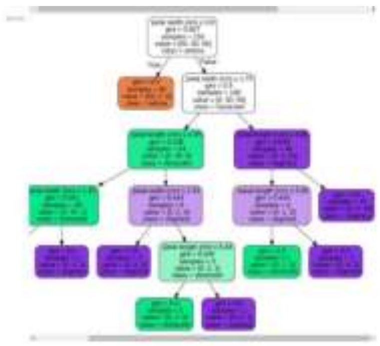
nformation gain is also called as Kullback-Leibler divergence denoted by IG(S,A) for a set S is the effective change in entropy after deciding on a particular attribute A. It measures the relative change in entropy with respect to the independent

variables. $IG(S,A) = H(S) - H(S,A)$ Alternatively, $IG(S,A) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$ where IG(S, A) is the information gain by applying feature A. H(S) is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A, where P(x) is the probability of event x.

PROGRAM:-

```
from sklearn.datasets import load_iris
from sklearn import tree
import graphviz
iris=load_iris()
X,y=iris.data,iris.target
clf=tree.DecisionTreeClassifier()
clf=clf.fit(X,y)
dot_data=tree.export_graphviz(clf,out_file=None,feature_names=iris.feature_names,class_names=iris.target_n
ames,filled=True,rounded=True,special_characters=True)
graph=graphviz.Source(dot_data)
graph.render("iris")
Graph
```

OUTPUT:-



Result:- Program Executed Successfully.

AIM:-Apply Naïve Bayes Classification algorithm on any dataset

DESC:-

- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

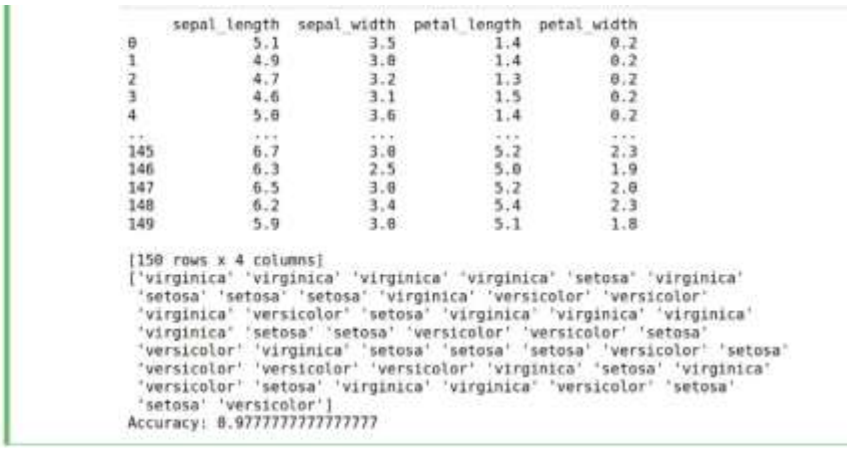
P(B) is Marginal Probability: Probability of Evidence.

PROGRAM:-

```
# importing required libraries
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# read the train and test dataset
train_data = pd.read_csv('train-data.csv')
test_data = pd.read_csv('test-data.csv')

# shape of the dataset
print('Shape of training data :',train_data.shape)
print('Shape of testing data :',test_data.shape)

# Now, we need to predict the missing target variable in the test data
# target variable - Survived

# seperate the independent and target variable on training data
train_x = train_data.drop(columns=['Survived'],axis=1)
train_y = train_data['Survived']

# seperate the independent and target variable on testing data
test_x = test_data.drop(columns=['Survived'],axis=1)
test_y = test_data['Survived']

'''
Create the object of the Naive Bayes model
You can also add other parameters and test your code here
Some parameters are : var_smoothing
Documentation of sklearn GaussianNB:

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

 '''
model = GaussianNB()
```

```
# fit the model with the training data
model.fit(train_x,train_y)

# predict the target on the train dataset
predict_train = model.predict(train_x)
print('Target on train data',predict_train)

# Accuray Score on train dataset
accuracy_train = accuracy_score(train_y,predict_train)
print('accuracy_score on train dataset : ', accuracy_train)
# predict the target on the test dataset
predict_test = model.predict(test_x)
print('Target on test data',predict_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(test_y,predict_test)
print('accuracy_score on test dataset : ', accuracy_test)
```

OUTPUT:-



```
     sepal_length  sepal_width  petal_length  petal_width
0             5.1          3.5           1.4          0.2
1             4.9          3.0           1.4          0.2
2             4.7          3.2           1.3          0.2
3             4.6          3.1           1.5          0.2
4             5.0          3.6           1.4          0.2
..            ...          ...           ...          ...
145           6.7          3.0           5.2          2.3
146           6.3          2.5           5.0          1.9
147           6.5          3.0           5.2          2.0
148           6.2          3.4           5.4          2.3
149           5.9          3.0           5.1          1.8

[150 rows x 4 columns]
['virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'virginica'
 'setosa' 'setosa' 'setosa' 'virginica' 'versicolor' 'versicolor'
 'virginica' 'versicolor' 'setosa' 'virginica' 'virginica' 'virginica'
 'virginica' 'setosa' 'setosa' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'setosa' 'setosa' 'setosa' 'versicolor' 'setosa'
 'versicolor' 'versicolor' 'versicolor' 'virginica' 'setosa' 'virginica'
 'versicolor' 'setosa' 'virginica' 'virginica' 'versicolor' 'setosa'
 'setosa' 'versicolor']
Accuracy: 0.9777777777777777
```

Result:- Program Executed Successfully.

AIM:-Generate frequent itemsets using Apriori Algorithm in python and also generate association rules for any market basket data.

DESC:-

The key concept in the Apriori algorithm is that it assumes all subsets of a frequent itemset to be frequent. Similarly, for any infrequent itemset, all its supersets must also be infrequent.**Grab high-paying analytics jobs with the help of these Top Data Science Interview Questions!**Let us try and understand the working of an Apriori algorithm with the help of a very famous business scenario, market basket analysis.

In order to select the interesting rules out of multiple possible rules from this small business scenario, we will be using the following measures:

- **Support**

$$\frac{\textit{Number of transactions in which the item wine appears}}{\textit{Total number of transactions}}$$

- **Confidence** $\frac{support(wine,chips,bread)}{support(wine,chips)}$

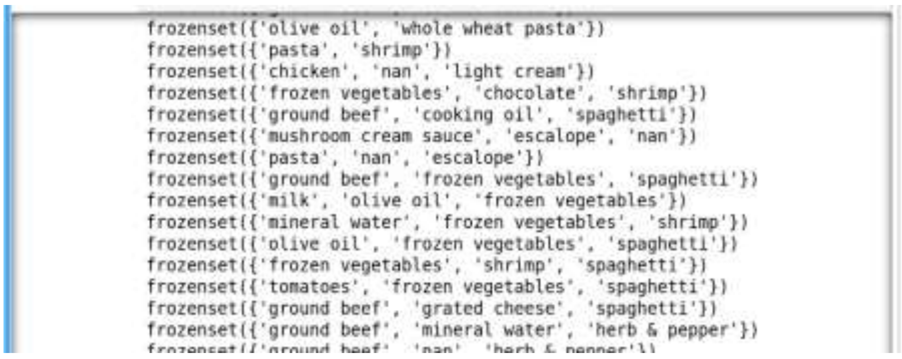
- **List**
- **Conviction**

$$conv(x \Rightarrow y) = \frac{1 - supp(y)}{1 - conf(x \Rightarrow y)}$$

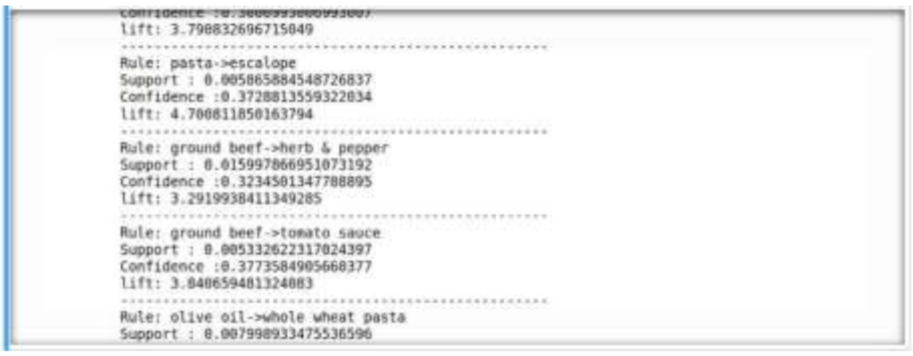
PROGRAM:-

```
import pandas as pd
from apyori import apriori
st_df=pd.read_csv("Market_Basket_Optimisation.csv",header=None)
st_df
#converting data frames into list of lists
l=[]
for i in range(1,7501):
   l.append([str(st_df.values[i,j]) for j in range(0,20)])
#Applying apriori algorithm
association_rules = apriori(l,min_support=0.0045, min_confidence=0.2 ,min_lift=3 ,min_length=2)
association_results = list(association_rules)
for i in range(0, len(association_results)):
   print(association_results[i][0])
for item in association_results:
   #first index of the inner list
   #contains base item and add item
   pair= item[0]
   items= [x for x in pair]
   print("Rule: " +items[0] + "->"+ items[1]  )
   #second index of the inner list
   print("Support: "+ str(item[1]))
   #third index of the list located at 0th position of the third index of the inner list
   print("Confidence: "+str(item[2][0][2]))
   print("Lift: " +str(item[2][0][3]))
   print("-------------------------------------------------------------")
```

OUTPUT:-



```
frozenset({'olive oil', 'whole wheat pasta'})
frozenset({'pasta', 'shrimp'})
frozenset({'chicken', 'nan', 'light cream'})
frozenset({'frozen vegetables', 'chocolate', 'shrimp'})
frozenset({'ground beef', 'cooking oil', 'spaghetti'})
frozenset({'mushroom cream sauce', 'escalope', 'nan'})
frozenset({'pasta', 'nan', 'escalope'})
frozenset({'ground beef', 'frozen vegetables', 'spaghetti'})
frozenset({'milk', 'olive oil', 'frozen vegetables'})
frozenset({'mineral water', 'frozen vegetables', 'shrimp'})
frozenset({'olive oil', 'frozen vegetables', 'spaghetti'})
frozenset({'frozen vegetables', 'shrimp', 'spaghetti'})
frozenset({'tomatoes', 'frozen vegetables', 'spaghetti'})
frozenset({'ground beef', 'grated cheese', 'spaghetti'})
frozenset({'ground beef', 'mineral water', 'herb & pepper'})
frozenset({'ground beef', 'nan', 'herb & pepper'})
```



```
lift: 3.790832696715049
-------------------------------------------------------
Rule: pasta->escalope
Support : 0.005865884548726837
Confidence :0.3728813559322034
lift: 4.700811850163794
-------------------------------------------------------
Rule: ground beef->herb & pepper
Support : 0.015997866951073192
Confidence :0.3234501347708895
lift: 3.2919938411349285
-------------------------------------------------------
Rule: ground beef->tomato sauce
Support : 0.005332622317024397
Confidence :0.3773584905660377
lift: 3.840659481324083
-------------------------------------------------------
Rule: olive oil->whole wheat pasta
Support : 0.007998933475536596
```

Result:- Program Executed Successfully.

AIM:-Apply K- Means clustering algorithm on any dataset.

DESC:-

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

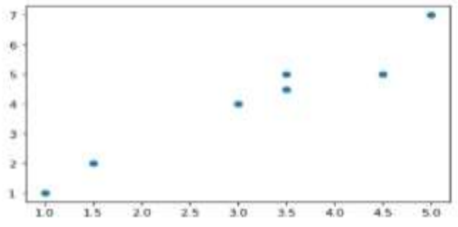
$$WCSS= \sum_{P_i \text{ in Cluster1}} distance(P_i\ C_1)^2 + \sum_{P_i \text{ in Cluster2}} distance(P_i\ C_2)^2 + \sum_{P_i \text{ in CLuster3}} distance(P_i\ C_3)^2$$

Program:-

```
import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import KMeans

X=np.array([[1,1],[1.5,2],[3,4],[5,7],[3.5,5],[4.5,5],[3.5,4.5]])

print(X)

KMeans=KMeans(n_clusters=2)

KMeans.fit(X)

plt.scatter(X[:,0],X[:,1])

print('\n\nclusters',KMeans.cluster_centers_)

print(KMeans.labels_)
```

Output:-

```
[[1.  1. ]
 [1.5 2. ]
 [3.  4. ]
 [5.  7. ]
 [3.5 5. ]
 [4.5 5. ]
 [3.5 4.5]]


clusters [[1.25 1.5 ]
 [3.9  5.1 ]]
[0 0 1 1 1 1 1]
```



Result:- Program Executed Successfully.

AIM:-Apply Hierarchical Clustering algorithm on any dataset.

DESC:-

Following are the steps involved in agglomerative clustering:

1. At the start, treat each data point as one cluster. Therefore, the number of clusters at the start will be K, while K is an integer representing the number of data points.
2. Form a cluster by joining the two closest data points resulting in K-1 clusters.
3. Form more clusters by joining the two closest clusters resulting in K-2 clusters.
4. Repeat the above three steps until one big cluster is formed.
5. Once single cluster is formed, dendrograms are used to divide into multiple clusters depending upon the problem. We will study the concept of dendrogram in detail in an upcoming section.

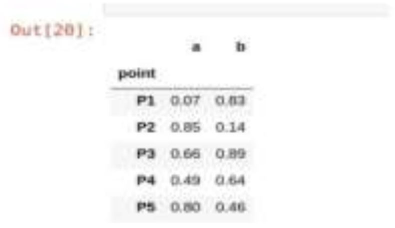
There are different ways to find distance between the clusters. The distance itself can be Euclidean or Manhattan distance. Following are some of the options to measure distance between two clusters:

1. Measure the distance between the closes points of two clusters.
2. Measure the distance between the farthest points of two clusters.
3. Measure the distance between the centroids of two clusters.
4. Measure the distance between all possible combination of points between the two clusters and take the mean.

PROGRAM:-

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as shc
from scipy.spatial.distance import squareform,pdist
a=[0.07,0.85,0.66,0.49,0.80]
b=[0.83,0.14,0.89,0.64,0.46]
point=['P1','P2','P3','P4','P5']
data=pd.DataFrame({'point':point,'a':np.round(a,2),'b':np.round(b,2)})
data=data.set_index('point')
data
```

OUTPUT:-



```
Out[20]:
          a     b
point
   P1  0.07  0.83
   P2  0.85  0.14
   P3  0.66  0.89
   P4  0.49  0.64
   P5  0.80  0.46
```

/

Result:- Program Executed Successfully.

AIM:-Apply DBSCAN clustering algorithm on any dataset.

DESC:-

The following are the DBSCAN clustering algorithmic steps:

- **Step 1:** Initially, the algorithms start by selecting a point (x) randomly from the data set and finding all the neighbor points within *Eps* from it. If the number of *Eps-neighbours* is greater than or equal to **MinPoints**, we consider x a core point. Then, with its *Eps-neighbours*, x forms the first cluster.After creating the first cluster, we examine all its member points and find their respective *Eps -neighbors*. If a member has at least *MinPoints Eps-neighbours*, we expand the initial cluster by adding those *Eps-neighbours* to the cluster. This continues until there are no more points to add to this cluster.

  Step 2: For any other core point not assigned to cluster, create a new cluster.

  Step 3: To the core point cluster, find and assign all points that are recursively connected to it.

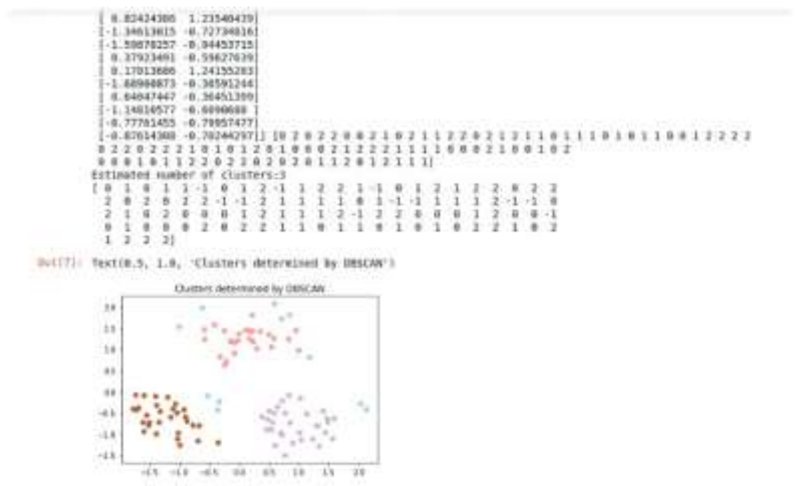
  Step 4: Iterate through all unattended points in the dataset and assign them to the nearest cluster at *Eps* distance from themselves. If a point does not fit any available clusters, locate it as a noise point.

PROGRAM:-

```
import numpy as np
from sklearn.datasets import make_blobs#generating a sequence of numbers
from sklearn.preprocessing import StandardScaler#preprocessing technique used to scale the graph
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
centers=[[0.5,2],[-1,-1],[1.5,-1]]


#scaling can be done by StandardScaler
#create dataset
X,y=make_blobs(n_samples=100, centers=centers, cluster_std=0.5,random_state=0)
#print(X,y)
X=StandardScaler().fit_transform(X)
print(X,y)
db=DBSCAN(eps=0.4,min_samples=5)
db.fit(X)
labels=db.labels_
n_clusters_=len(set(labels))-(1 if -1 in labels else 0)
print("Estimated number of clusters:%d" %n_clusters_)
y_pred=db.fit_predict(X)
```

```
print(db.labels_)
plt.figure(figsize=(6,4))
plt.scatter(X[:,0],X[:,1],c=y_pred ,cmap='Paired')
plt.title("Clusters determined by DBSCAN")
```

OUTPUT:-



```
 [ 0.82424386  1.23540439]
 [-1.34613615 -0.72734816]
 [-1.59878257 -0.94453715]
 [ 0.37923491 -0.59627639]
 [ 0.17013686  1.24155283]
 [-1.68988873 -0.38591244]
 [ 0.64047447 -0.36451399]
 [-1.14810577 -0.6090688 ]
 [-0.77781455 -0.79957477]
 [-0.87614388 -0.70244297]] [0 2 0 2 2 0 0 2 1 0 2 1 1 2 2 0 2 1 2 1 1 0 1 1 1 0 1 0 1 1 0 0 1 2 2 2 2
 0 2 2 0 2 2 2 1 0 1 0 1 2 0 1 0 0 0 2 1 2 2 2 1 1 1 1 0 0 0 2 1 0 0 1 0 2
 0 0 0 1 0 1 1 2 2 0 2 2 0 2 0 2 0 1 1 2 0 1 2 1 1 1]
Estimated number of clusters:3
[ 0  1  0  1  1 -1  0  1  2 -1  1  2  2  1 -1  0  1  2  1  2  2  0  2  2
  2  0  2  0  2  2 -1 -1  2  1  1  1  1  0  1 -1 -1  1  1  1  2 -1 -1  0
  2  1  0  2  0  0  0  1  2  1  1  1  2 -1  2  2  0  0  0  1  2  0  0 -1
  0  1  0  0  0  2  0  2  2  1  1  0  1  1  0  1  0  1  0  2  2  1  0  2
  1  2  2  2]

Out[7]: Text(0.5, 1.0, 'Clusters determined by DBSCAN')
```

Result:- Program Executed Successfully.