LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

Department of Computer Science & Engineering

20CS58 Data Mining Using Python Lab

Name of the Student: Kathoju Manoj

Registered Number: 21761A05G0

Branch & Section: CSE **&** C **../Sec**

Academic Year: **2022– 23**

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

CERTIFICATE

Certificate that this is a bonafied record of the practical work

done in ____Laboratory by

__________________________ with Regd. No. _______________________ of

______________B.Tech Course________ _________Semester in

____ Branch during the

Academic Year 2022-23

No. of Experiments held: ________

No. of Experiments Done: ________

Date: ..... / ____/ 2023 **Signature of the Faculty**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Name: Kathoju Manoj **LAB : Data Mining using python**
Class: CSE-C SEC **TASK No : 01**
Roll No: 21761A05G0 **Date : ________________**

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

a) Loading the dataset
b) Identifying the dependent and independent variables.
c) Dealing with missing data

Description/Theory:

A dataset is a collection of data that is used to train the model. A dataset acts as an example to teach the machine learning algorithm how to make predictions.

The values of this variable *depend* on other variables. It is the outcome that you're studying. It's also known as the response variable, outcome variable, and left-hand variable. Statisticians commonly denote them using a Y. Traditionally, graphs place dependent variables on the vertical, or Y, axis.

Dependent Variables:

The values of this variable *depend* on other variables. It is the outcome that you're studying. It's also known as the response variable, outcome variable, and left-hand variable. Statisticians commonly denote them using a Y. Traditionally, graphs place dependent variables on the vertical, or Y, axis.

Independent Variables:

The name helps you understand their role in statistical analysis. These variables are *independent*. In this context, independent indicates that they stand alone and other variables in the model do not influence them. The researchers are not seeking to understand what causes the independent variables to change.

Missing Values:

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.In Pandas, usually, missing values are represented by NaN. It stands for Not a Number.

Handling Missing Values:

- Deleting the Missing Values.
- Imputing the Missing Values

Deleting the Missing Values:

This approach is not recommended. It is one of the quick and dirty techniques one can use to deal with missing values. If the missing value is of the type Missing Not At Random (MNAR), then it should not be deleted.

Imputing Missing Values:

This approach focus on replacing the missing data with mean or most frequent value in that feature.

PROCEDURE/CODE:

```
#a) Loading Data Set
import pandas as pd
data=pd.read_csv(r"D:\New folder\emp.csv")
data
```

Output:

	Country	Gender	Age	Salary	Purchased
0	France	Male	44.0	72000.0	No
1	Spain	Female	27.0	48000.0	Yes
2	Germany	Male	30.0	54000.0	No

3	Spain	Male	38.0	61000.0	No
4	Germany	Female	40.0	NaN	Yes
5	France	Female	35.0	58000.0	Yes
6	Spain	Female	NaN	52000.0	No
7	France	Male	48.0	79000.0	Yes
8	Germany	Male	50.0	83000.0	No
9	France	Male	37.0	67000.0	Yes

#b) Dependent & Independent

```
import pandas as pd
data=pd.read_csv(r"D:\New folder\emp.csv")
x=data.iloc[:,:-1].values
y=data.iloc[:,-1].values
print(x)
print(y)
```

Output:

```
[['France' 'Male' 44.0 72000.0]
 ['Spain' 'Female' 27.0 48000.0]
 ['Germany' 'Male' 30.0 54000.0]
 ['Spain' 'MAle' 38.0 61000.0]
 ['Germany' 'Female' 40.0 nan]
 ['France' 'Female' 35.0 58000.0]
 ['Spain' 'Female' nan 52000.0]
 ['France' 'Male' 48.0 79000.0]
 ['Germany' 'Male' 50.0 83000.0]
 ['France' 'Male' 37.0 67000.0]]
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

```
#c) Handling missing Data
import numpy as np

# Importing the SimpleImputer class
from sklearn.impute import SimpleImputer

# Imputer object using the mean strategy and
# missing_values type for imputation
imputer = SimpleImputer(missing_values = np.nan,   strategy ='mean')
data = [[12, np.nan, 34], [10, 32, np.nan],  [np.nan, 11, 20]]

print("Original Data : \n", data)
# Fitting the data to the imputer object
imputer = imputer.fit(data)

# Imputing the data
data = imputer.transform(data)

print("Imputed Data : \n", data)
```

Output:

```
Original Data :
 [[12, nan, 34], [10, 32, nan], [nan, 11, 20]]
Imputed Data :
 [[12.  21.5 34. ]
 [10.  32.  27. ]
 [11.  11.  20. ]]
```

RESULT: The programs loading dataset , identifying dependent and Independent and for dealing with missing data is successfully executed.

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

a) Dealing with categorical data.
b) Scaling the features.
c) Splitting dataset into Training and Testing Sets.

Description/Theory :

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

1.Dealing with the Categorical Data:

Categorical data:

Categorical Data is the statistical data comprising categorical variables of data that are converted into categories.

One-hot Encoding:

In this method, each category is mapped to a vector that contains 1 and 0 denoting the presence or absence of the feature. The number of vectors depends on the number of categories for features.

Label Encoder

In label encoding, each category is assigned a value from 1 through N where N is the number of categories for the feature. There is no relation or order between these assignments.

2.Featuring Scale:

When your data has different values, and even different measurement units, it can be difficult to compare them. What is kilograms compared to meters? Or altitude compared to time?The answer to this problem is scaling. We can scale data into new values that are easier to compare.

3.Training Data:

The training data is the biggest (in -size) subset of the original dataset, which is used to train or fit the machine learning model. Firstly, the training data is fed to the ML algorithms, which lets them learn how to make predictions for the given task.

Testing Data:

This dataset evaluates the performance of the model and ensures that the model can generalize well with the new or unseen dataset. *The test dataset is another subset of original data, which is independent of the training dataset*.

Splitting:

For splitting the dataset, we can use the train_test_split function of scikit-learn.

PROCEDURE/CODE:

a) Dealing with categorical data

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[3])],remainder='passthrough')
x=np.array(ct.fit_transform(x))
print(x)

import pandas as pd
from sklearn import preprocessing
df=pd.read_csv(r"D:\New folder\iris.csv")
# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
```

```
df['variety']= label_encoder.fit_transform(df['variety'])

Df['variety']
```

Output

```
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]
 [0.0 0.0 1.0 131876.9 99814.71 362861.36]]

0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: variety, Length: 150, dtype: int32
```

b) Scaling the features

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
df = pandas.read_csv("data.csv")
X = df[['Weight', 'Volume']]
scaledX = scale.fit_transform(X)
print(scaledX)
```

Output

```
[[-2.10389253 -1.59336644]
 [-0.55407235 -1.07190106]
 [-1.52166278 -1.59336644]
 [-1.78973979 -1.85409913]
 [-0.63784641 -0.28970299]
 [ 0.15800719 -0.0289703 ]
 [ 0.3046118  -0.0289703 ]
 [-0.05142797  1.53542584]
 [-0.72580918 -0.0289703 ]]
```

c) Splitting dataset into Training and Testing Sets.

```
import numpy as np
from sklearn.model_selection import train_test_split
#splitting the data into training and  testing
x = np.arange(1, 25).reshape(12, 2)
y = np.array([0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0])
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
print(x_train)
print(x_test)
print(y_train)
print(x_test)
```

Output:

```
[[ 3  4]
 [13 14]
 [ 1  2]
 [15 16]
 [23 24]
 [19 20]
 [17 18]
 [11 12]]
[[ 5  6]
 [ 7  8]
 [ 9 10]
 [21 22]]
[1 0 0 1 0 0 1 0]
[[ 5  6]
 [ 7  8]
 [ 9 10]
 [21 22]]
```

RESULT: The programs for data pre-processing , feature scaling and splitting dataset is successfully executed.

AIM: Demonstrate the following Similarity and Dissimilarity Measures using python

a) Pearson's Correlation
b) Cosine Similarity
c) Jaccard Similarity
d) Euclidean Distance
e) Manhattan Distance

Description/Theory:

Similarity Measure:

Measures of similarity provide a numerical value which indicates the strength of associations between objects or variables. The extent to which the variables are corresponding with each other is usually indicated between "0" and "1" where "0" means no similarity or exclusion and "1" means perfect similarity of identity.

Dissimilarity Measure:

Numerical measure of how different two data objects are.Range from 0 (objects are alike) to ∞ (objects are different).

Pearsons coefficient:

The Pearson coefficient is a type of correlation coefficient that represents the relationship between two variables that are measured on the same interval or ratio scale. The Pearson coefficient is a measure of the strength of the association between two continuous variables.

Cosine similarity:

Cosine similarity is a metric used to measure the similarity of two vectors. Specifically, it measures the similarity in the direction or orientation of the vectors ignoring differences in their magnitude or scale. Both vectors need to be part of the same inner product space, meaning they must produce a scalar through inner product multiplication. The similarity of two vectors is measured by the cosine of the angle between them.

Jaccard Similarity is a common proximity measurement used to compute the similarity between two objects, such as two text documents. Jaccard similarity can be used to find the similarity between two asymmetric binary vectors

Euclidean:

Euclidean distance Similarity is a common proximity measurement used to compute the similarity between two objects, such as two arrays. Euclidean distance similarity can be used to find the similarity between two asymmetric binary vectors.

Minkowski :

Distance calculates the distance between two real-valued vectors. It is a generalization of the Euclidean and Manhattan distance measures and adds a parameter, called the "order" or "p", that allows different distance measures to be calculated.

PROCEDURE/CODE:

a) Pearson

```
 #importing libraries
from scipy.stats import pearsonr
#Creating two lists.
x=[-2,-1,0,1,2]
y=[4,1,3,2,0]
corr=pearsonr(x,y)
print('pearson correlation: ',corr)
```

Output:

```
pearson correlation:  PearsonRResult(statistic=-0.7000000000000001,
pvalue=0.1881204043741873)
```

b) Cosine

```
 import numpy as np
from numpy.linalg import norm

#defining two arrays
A=np.array([2,1,2,3,3,9])
B=np.array([3,4,2,4,5,5])
print("A: ",A)
print("B: ",B)

#compute ccosine similarity
cosine=np.dot(A,B)/norm(A)*norm(B)
print("cosine Similarity :",cosine)
```

Output

```
A:  [2 1 2 3 3 9]
B:  [3 4 2 4 5 5]
    cosine Similarity : 80.6581721881964
```

c) Jaccards

```
import numpy as np
from scipy.spatial.distance import jaccard
A=np.array([1,0,0,1,1,1])
B=np.array([0,0,1,1,1,1])

#Calculating jaccard similarity

d=jaccard(A,B)
print("distance:",d)
```

Output

```
distance: 0.4
```

d) Euclidean

```
from sklearn.metrics.pairwise import euclidean_distances
x=[[0,1],[1,1]]
```

```
euclidean_distances(x,x)

#calculating euclidean distance between two vectors
from scipy.spatial.distance import euclidean
row1=[10,20,15,10,5]
row2=[12,24,18,8,7]
dist=euclidean(row1,row2)
print(dist)
```

Output

```
6.082762530298219
```

e) Manhattan

```
 #importing the libraries
from scipy.spatial import manhattan_distance
x1 = (1,2,3,4,5,6)
x2=(10,20,30,1,2,3)
#calculating manhattan distance
print(manhattan_distance(x1, x2))
```

Output:

```
63
```

RESULT:
The program for finding similarity and dissimilarity for above measures is successfully executed.

Name: Kathoju Manoj **LAB : Data Mining using python**
Class: CSE-C SEC **TASK No : 04**
Roll No: 21761A05G0 **Date : ________________**

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

Description/Theory: Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

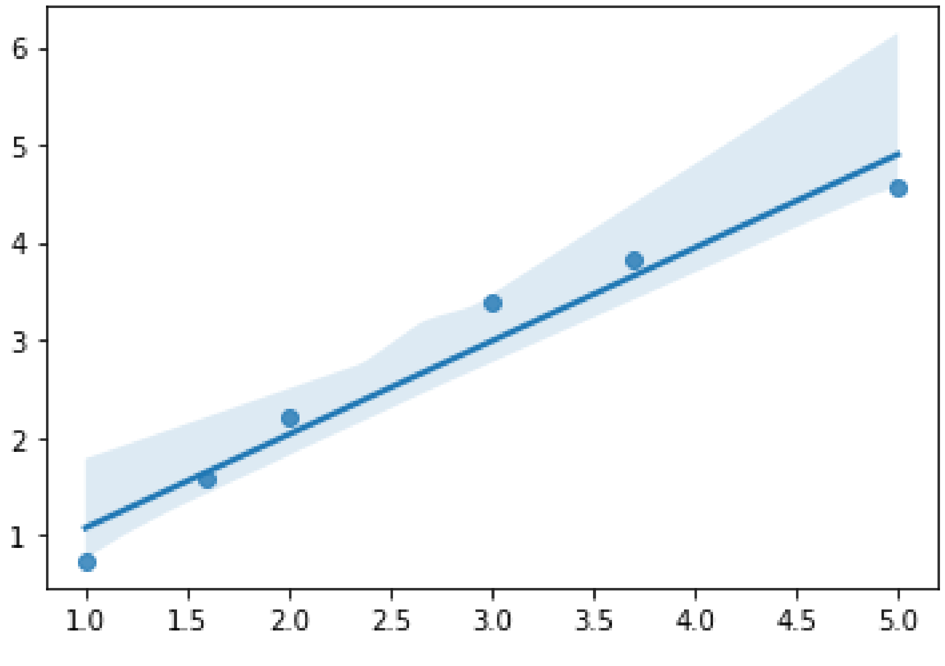
PROCEDURE/CODE:

```
from sklearn import linear_model
import pandas as pd
import numpy as np
import seaborn as sb
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error
data=pd.read_csv("/travel_petrol.csv")
print(data)
mt=data['travel'].mean()
mp=data['petrol'].mean()
print(mt,mp)
b1=data.cov()
xx = data['travel'].values.reshape((-1,1))
yy = data['petrol'].values.reshape((-1,1))
reg=linear_model.LinearRegression()
reg.fit(xx,yy)
pre=reg.predict(xx)
print(reg.intercept_)
print(reg.coef_)
print('score r2 ',reg.score(xx,yy))
print(r2_score(pre,yy))
print("mse",mean_squared_error(pre,yy))
print(pre)
#var=data.var(data['travel'])
print(b1)
s=0
t=0
for i in pre:
  s=(i[0]-mp)**2+s
for i in yy:
  t=(i[0]-mp)**2+t
print(' r ^2 is :',s/t)
```

```
sb.regplot(yy,pre)
```

Output:

```
travel  petrol
0       20      1.0
1       45      3.0
2       56      5.0
3       34      2.0
4       28      1.6
5       49      3.7
38.666666666666664 2.716666666666667
[-1.38664996]
[[0.10612026]]
score r2  0.956227211870097
0.9542234444005244
mse 0.08001908852302554
[[0.73575519]
 [3.38876163]
 [4.55608447]
 [2.2214388 ]
 [1.58471725]
 [3.81324266]]
            travel     petrol
travel  186.266667  19.766667
petrol   19.766667   2.193667
 r ^2 is : 0.9562272118700975
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: 
Pass the following variables as keyword args: x, y. From version 0.12, the only 
valid positional argument will be `data`, and passing other arguments without an 
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<AxesSubplot:>
```



RESULT: linear regression algorithm on petrol.csv dataset successfully executed.

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

Description/Theory:

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**

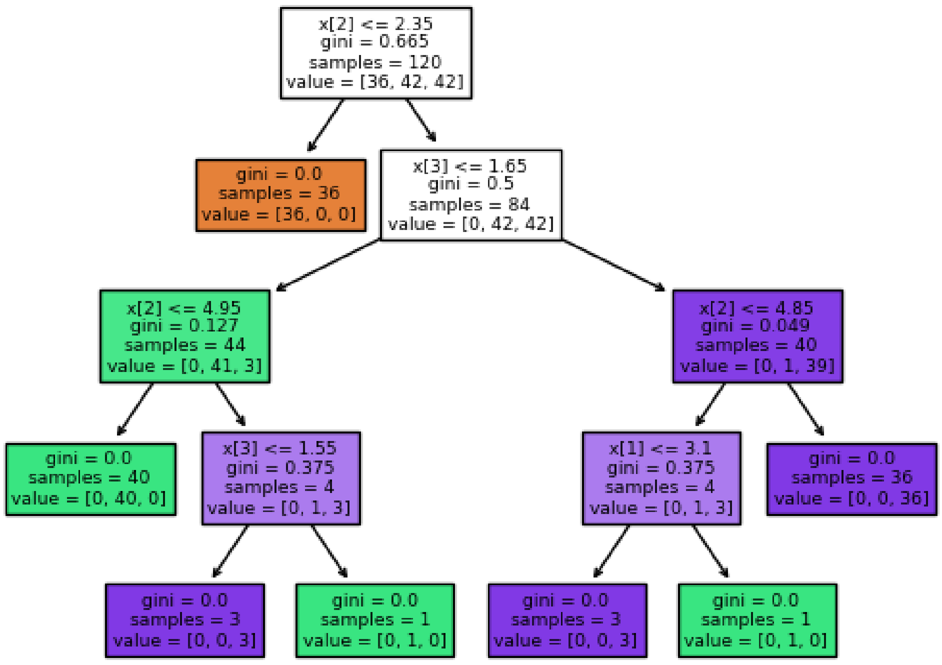
PROCEDURE/CODE:

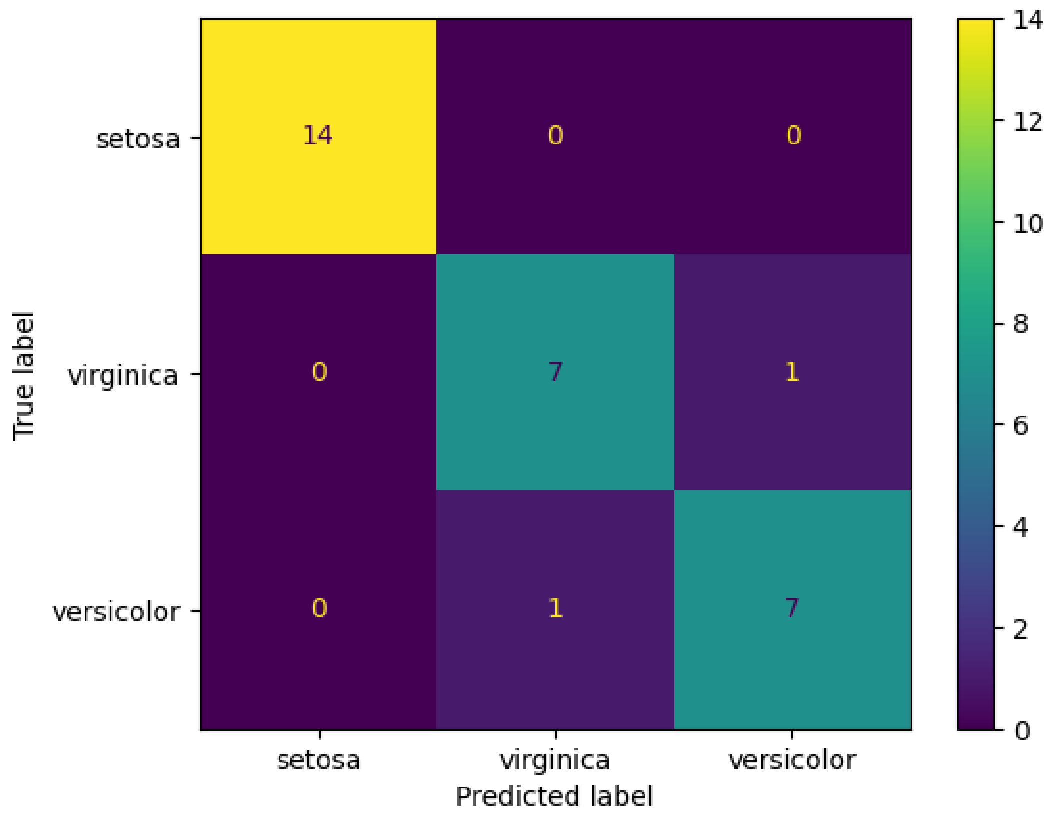
```
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import
confusion_matrix,ConfusionMatrixDisplay,precision_score,recall_score,f
1_score
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np

data=sb.load_dataset('iris')

X=data.iloc[:,:4]
Y=data.iloc[:,-1]

xtrain,xtest,ytrain,ytest =
train_test_split(X,Y,test_size=0.2,random_state=2)

model=DecisionTreeClassifier()

model=model.fit(xtrain,ytrain)

ypre=model.predict(xtest)
plot_tree(model,filled=True)

print("accuaracy score :",accuracy_score(ytest,ypre))
cm=confusion_matrix(ytest,ypre)
print("confusion matrix :\n",confusion_matrix(ytest,ypre))
```

```
xtestdata=[8,31,4.7,8.3]
xtestdata=np.array(xtestdata).reshape(1,-1)
yout=model.predict(xtestdata)
print('output ',yout)
dis=ConfusionMatrixDisplay(confusion_matrix=cm, display_labels =
['setosa','virginica','versicolor'])
dis.plot()
plt.show()

print("f1 score :",f1_score(ytest,ypre,average='weighted'))

print(classification_report(ytest,ypre))

#plt.scatter(xtest,ytest)
```

Output:

```
accuaracy score : 0.9333333333333333
confusion matrix :
 [[14  0  0]
 [ 0  7  1]
 [ 0  1  7]]
output  ['versicolor']
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
have valid feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```





```
f1 score : 0.9333333333333333
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       0.88      0.88      0.88         8
   virginica       0.88      0.88      0.88         8

    accuracy                           0.93        30
   macro avg       0.92      0.92      0.92        30
weighted avg       0.93      0.93      0.93        30
```

RESULT: Decision Tree Classification algorithm successfully implemented.

AIM: Apply Naïve Bayes Classification algorithm on any dataset.

Description/Theory:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.

It is mainly used in *text classification* that includes a high-dimensional training dataset.

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.

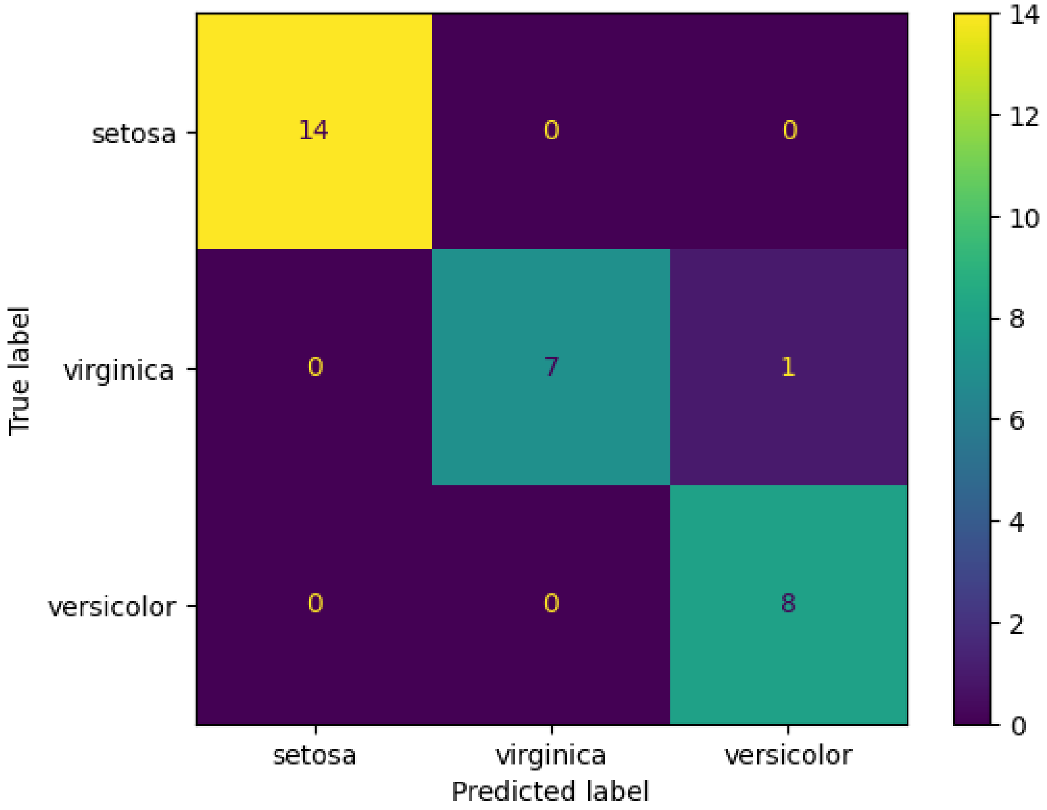
PROCEDURE/CODE:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import
confusion_matrix,ConfusionMatrixDisplay,precision_score,recall_score,f
1_score
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
data=sb.load_dataset('iris')

X=data.iloc[:,:4]
Y=data.iloc[:,-1]

xtrain,xtest,ytrain,ytest =
train_test_split(X,Y,test_size=0.2,random_state=2)

model=GaussianNB()

model=model.fit(xtrain,ytrain)

ypre=model.predict(xtest)

print("accuaracy score :",accuracy_score(ytest,ypre))
cm=confusion_matrix(ytest,ypre)
print("confusion matrix :\n",confusion_matrix(ytest,ypre))
```

```
xtestdata=[8,31,4.7,8.3]
xtestdata=np.array(xtestdata).reshape(1,-1)
yout=model.predict(xtestdata)
print('output ',yout)
dis=ConfusionMatrixDisplay(confusion_matrix=cm, display_labels =
['setosa','virginica','versicolor'])
dis.plot()
plt.show()
print("f1 score :",f1_score(ytest,ypre,average='weighted'))
print(classification_report(ytest,ypre))
```

Output:

```
accuaracy score : 0.9666666666666667
confusion matrix :
 [[14  0  0]
 [ 0  7  1]
 [ 0  0  8]]
output  ['virginica']
```



```
f1 score : 0.9665359477124185
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       1.00      0.88      0.93         8
   virginica       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.96      0.96        30
weighted avg       0.97      0.97      0.97        3
```

RESULT: Naïve Bayes Classification algorithm successfully implemented.

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

Description/Theory:

Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected.

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable.

PROCEDURE/CODE:

```
import pandas as pd
import numpy as np
from apyori import apriori

data=pd.read_csv('store_data.csv')
# display(data.head(10))
# data=data.iloc[:50,:]
print(data.shape)
records=[]
# print(data.values[50,19])
for i in range(0,7500):
  records.append([str(data.values[i,j]) for j in range(0,20)])
print(len(records))

ar=apriori(records,min_support=0.0045,min_confidence=0.2,min_lift=3,mi
n_length=2,max_length=2)
li=list(ar)
for i in li:
  pair=i[0]
  tt=[x for x in pair]
  print("rule :"+tt[0]+"-->"+tt[1])
  print('support :'+str([i[1]]))
  print('confidence :'+str(i[2][0][2]))
  print('Lift :'+str(i[2][0][3]))
  print('-------------------------------------------------')
```

Output:

```
(7500, 20)
7500
rule :light cream-->chicken
support :[0.004533333333333334]
confidence :0.2905982905982906
Lift :4.843304843304844
```

```
-------------------------------------------------
rule :mushroom cream sauce-->escalope
support :[0.005733333333333333]
confidence :0.30069930069930073
Lift :3.7903273197390845
-------------------------------------------------
rule :pasta-->escalope
support :[0.005866666666666667]
confidence :0.37288135593220345
Lift :4.700185158809287
-------------------------------------------------
rule :ground beef-->herb & pepper
support :[0.016]
confidence :0.3234501347708895
Lift :3.2915549671393096
-------------------------------------------------
rule :ground beef-->tomato sauce
support :[0.005333333333333333]
confidence :0.37735849056603776
Lift :3.840147461662528
-------------------------------------------------
rule :whole wheat pasta-->olive oil
support :[0.008]
confidence :0.2714932126696833
Lift :4.130221288078346
-------------------------------------------------
rule :shrimp-->pasta
support :[0.005066666666666666]
confidence :0.3220338983050848
Lift :4.514493901473151
-------------------------------------------------
```

RESULT: Frequent itemsets using Apriori Algorithm and association rules for any market basket data successfully implemented.

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

Description/Theory:

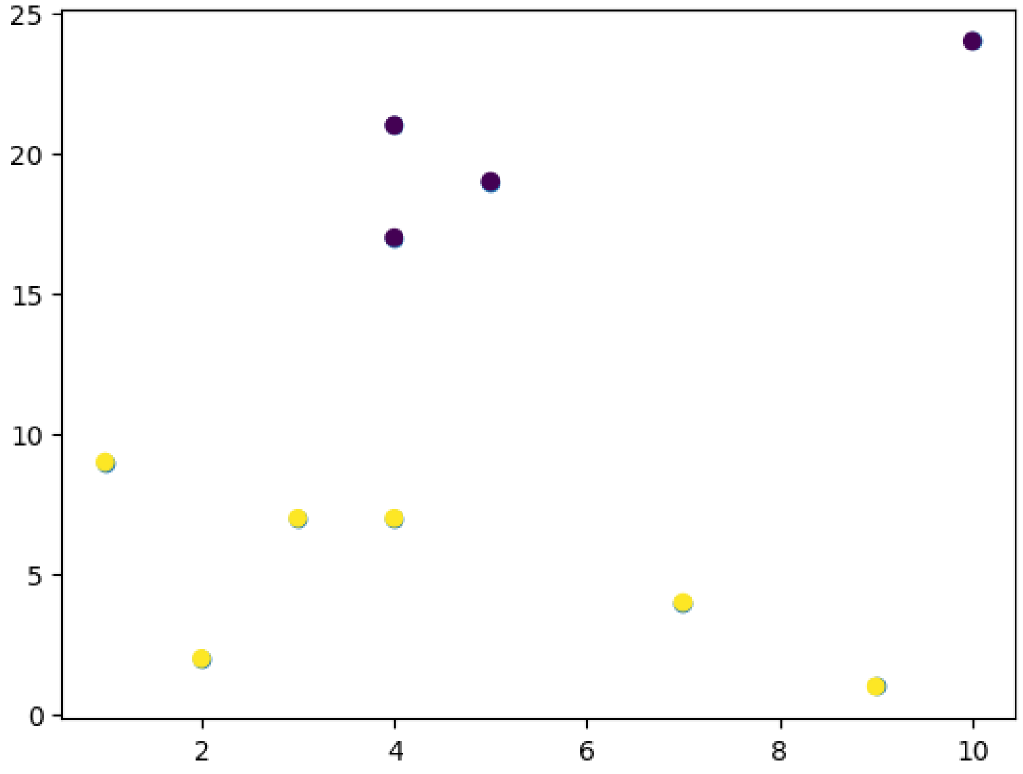
K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

PROCEDURE/CODE:

```
 #CLustering by k-means algorithm
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
x = [4, 5, 10, 4,3,7,1,2,9,4]
y = [21, 19, 24,17,7,4,9,2,1,7]
data=np.array(list(zip(x,y)))
plt.scatter(data[:,0],data[:,1])
model=KMeans(n_clusters=2)
model.fit(data)
plt.scatter(data[:,0],data[:,1], c=model.labels_)
plt.show()
print("\nClusters :",model.cluster_centers_)
print(model.labels_)
```

Output:

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4.
Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



```
Clusters : [[ 5.75       20.25      ]
 [ 4.33333333  5.        ]]
[0 0 0 0 1 1 1 1 1 1]
```

RESULT: Hence , KMeans algorithm successfully implemented.

AIM: Apply Hierarchical Clustering algorithm on any dataset.

Description/Theory:

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.

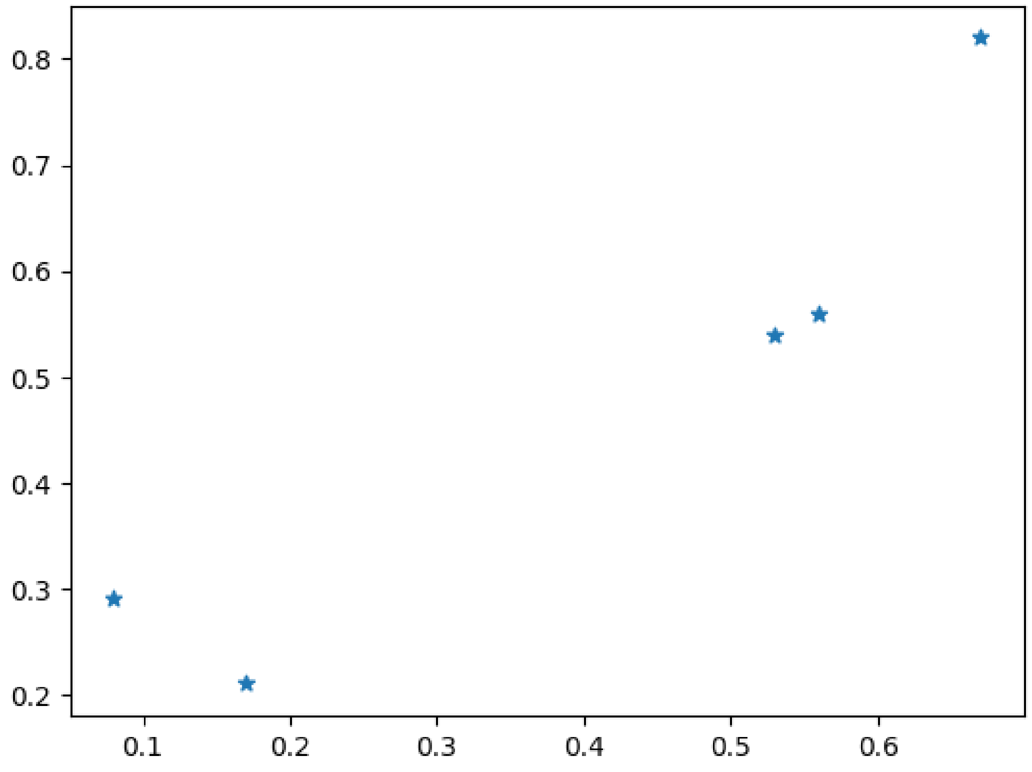
In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.

PROCEDURE/CODE:

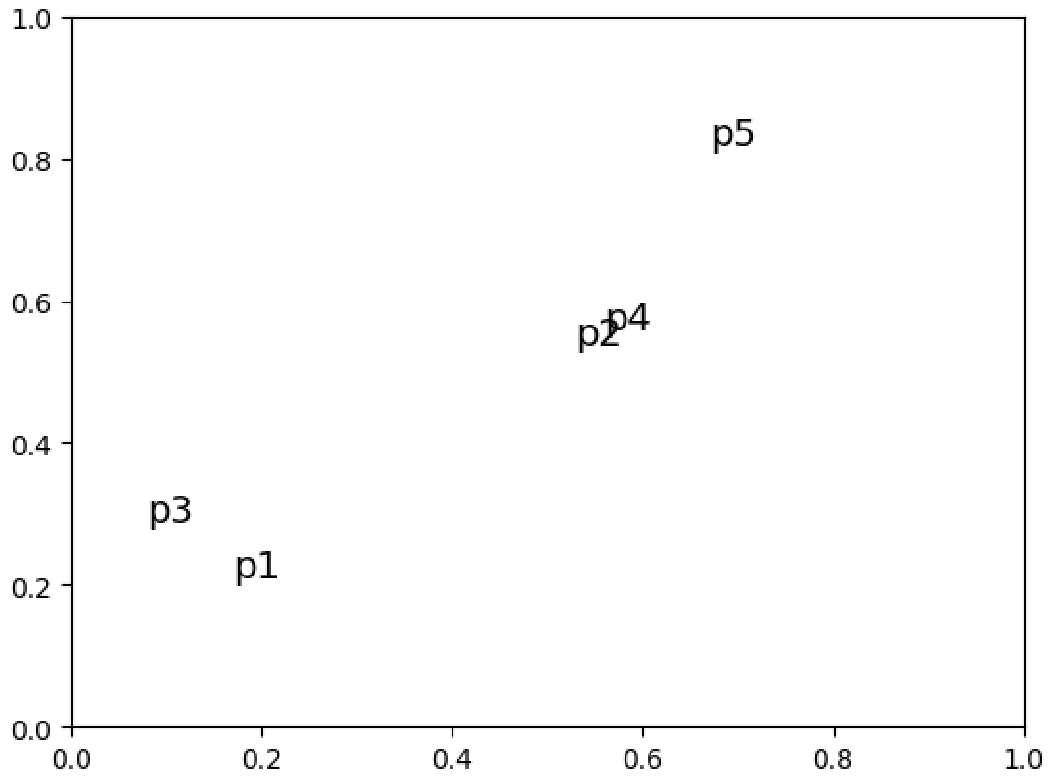
```
#hierachical Clustering

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from scipy.spatial.distance import squareform,pdist

a=np.random.random_sample(size=5)
b=np.random.random_sample(size=5)
point=['p1','p2','p3','p4','p5']
data=pd.DataFrame({
    'Point':point,'Cola':np.round(a,2),'Colb':np.round(b,2)
} )
data=data.set_index('Point')

print(data)
plt.scatter(data['Cola'],data['Colb'],marker='*')
plt.show()
for j in data.itertuples():
  plt.annotate(j.Index,(j.Cola,j.Colb),fontsize=14)
#DIstance Matrix
dist=pd.DataFrame(squareform(pdist(data[['Cola','Colb']]),'euclidean')
,columns=data.index.values,index=data.index.values)
print("Distance Matrix :")
print(dist)

plt.figure(figsize=(12,5))
plt.title('Dendrogram with single Linkage :)')
de=sch.dendrogram(sch.linkage(data[['Cola','Colb']],method='single'),l
abels=data.index)
from sklearn.cluster import AgglomerativeClustering
clu=AgglomerativeClustering(n_clusters=2,affinity='euclidean',linkage=
'single')
lol=clu.fit_predict(data)
sch.dendrogram(sch.linkage(data[['Cola','Colb']],method='single'),labe
ls=data.index)
```

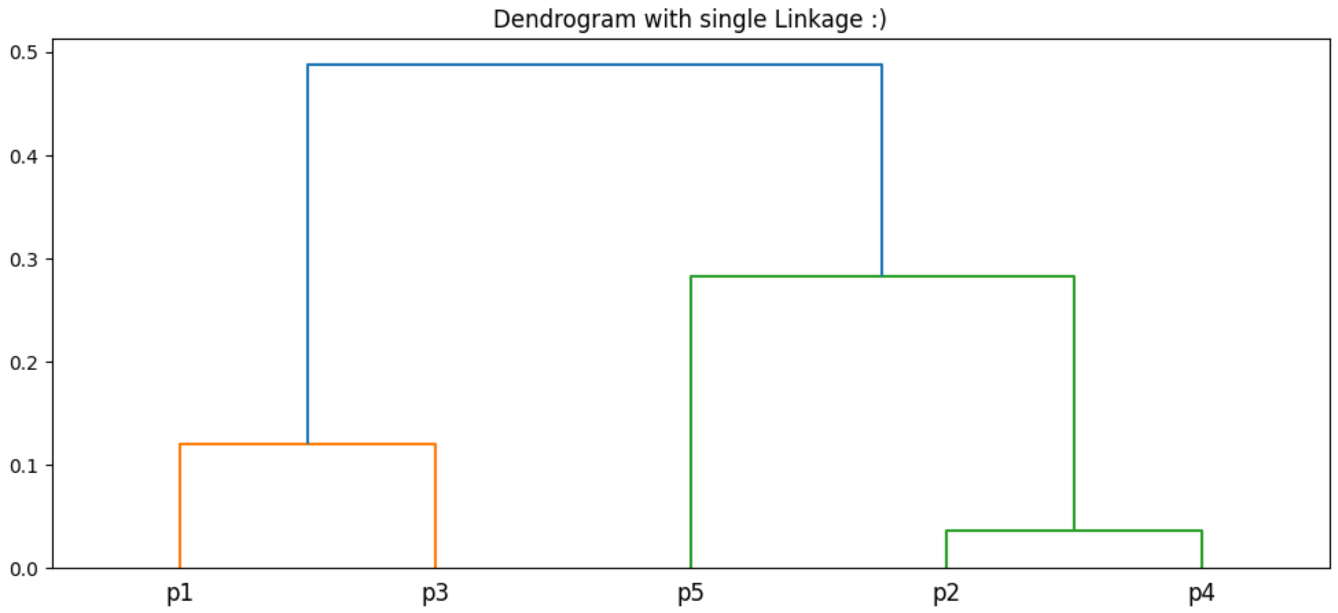
Output:

```
        Cola  Colb
Point
p1      0.17  0.21
p2      0.53  0.54
p3      0.08  0.29
p4      0.56  0.56
p5      0.67  0.82
```

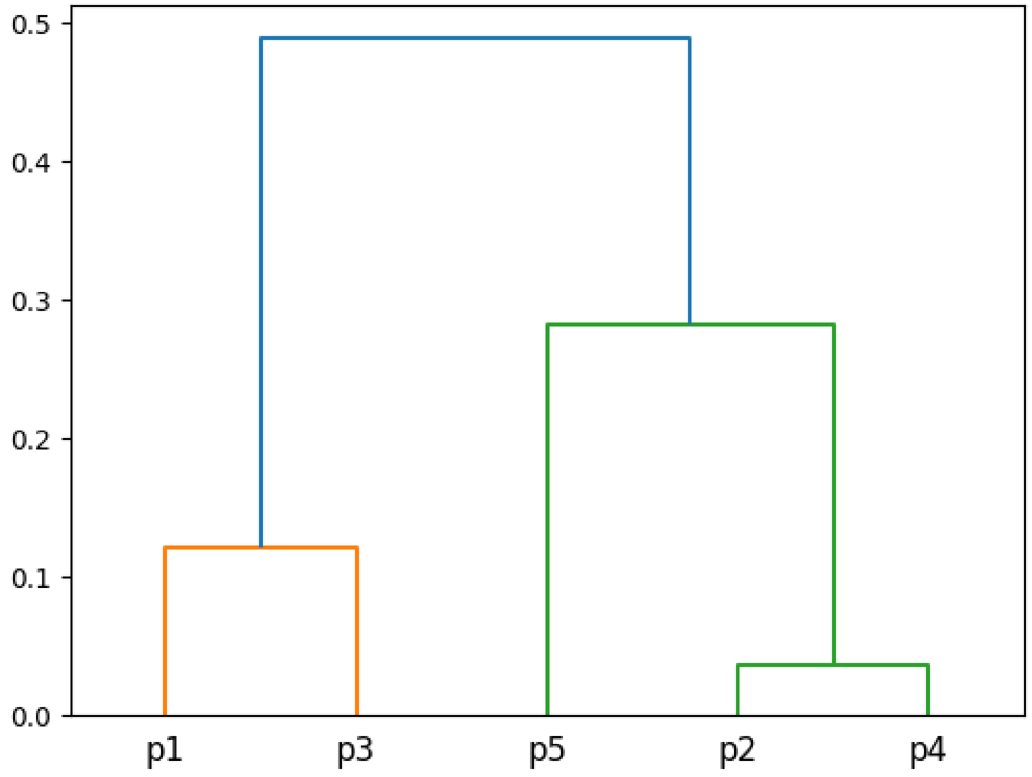


```
Distance Matrix :
          p1        p2        p3        p4        p5
p1  0.000000  0.488365  0.120416  0.524023  0.788733
p2  0.488365  0.000000  0.514782  0.036056  0.313050
p3  0.120416  0.514782  0.000000  0.550727  0.793095
p4  0.524023  0.036056  0.550727  0.000000  0.282312
p5  0.788733  0.313050  0.793095  0.282312  0.000000
```





```
{'icoord': [[5.0, 5.0, 15.0, 15.0],
  [35.0, 35.0, 45.0, 45.0],
  [25.0, 25.0, 40.0, 40.0],
  [10.0, 10.0, 32.5, 32.5]],
 'dcoord': [[0.0, 0.12041594578792295, 0.12041594578792295, 0.0],
  [0.0, 0.036055512754639925, 0.036055512754639925, 0.0],
  [0.0, 0.282311884269862, 0.282311884269862, 0.036055512754639925],
  [0.12041594578792295,
   0.48836461788299124,
   0.48836461788299124,
   0.282311884269862]],
 'ivl': ['p1', 'p3', 'p5', 'p2', 'p4'],
 'leaves': [0, 2, 4, 1, 3],
 'color_list': ['C1', 'C2', 'C2', 'C0'],
 'leaves_color_list': ['C1', 'C1', 'C2', 'C2', 'C2']}
```



RESULT: Hierarchical Clustering algorithm successfully implemented.

AIM: Apply DBSCAN clustering algorithm on any dataset.

Description/Theory:

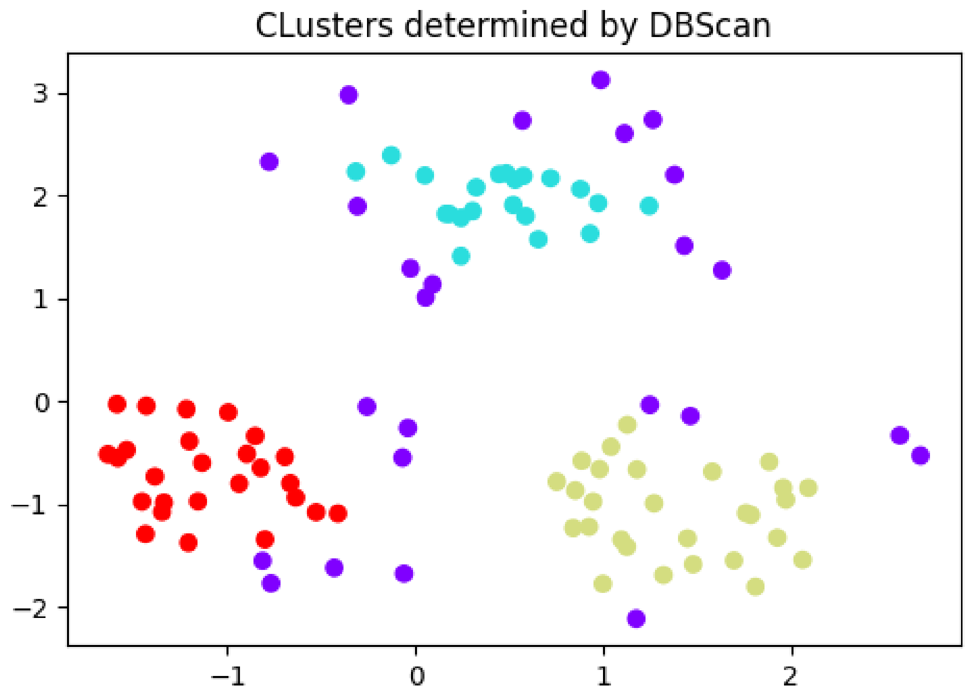
Clusters are dense regions in the data space, separated by regions of the lower density of points. The ***DBSCAN algorithm*** is based on this intuitive notion of "clusters" and "noise". The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.

PROCEDURE/CODE:

```
#DBScan

import numpy as np
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN

cen =[[0.5,2],[-1,-1],[1.5,-1]]

x,y=make_blobs(n_samples=100,centers=cen,cluster_std=0.5,random_state=
0)

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("Estimate cluster are %d"%(n_clusters_))
ypre=db.fit_predict(x)
print(db.labels_)
plt.figure(figsize=(6,4))
plt.scatter(x[:,0],x[:,1],c=ypre,cmap='rainbow')
plt.title("CLusters determined by DBScan")
plt.show()
```

Output:

```
Estimate cluster are 3
[ 0 -1  0  1  1 -1  0  1  2 -1  1  2  2  1 -1 -1 -1  2  1  2  2  0  2  2
  2  0  2 -1  2 -1 -1 -1  2  1  1  1  1 -1  1 -1 -1  1  1  1  2 -1 -1  0
  2  1  0 -1  0 -1  0  1 -1  1  1  1  2 -1  2  2  0  0  0  1  2 -1  0 -1
  0  1  0 -1  0  2 -1 -1  2  1  1  0  1  1  0 -1  0  1  0  2  2  1  0  2
  1  2  2  2]
```



RESULT: DBSCAN Algorithm successfully implemented.