LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING
(AUTONOMOUS)

Data Mining using Python Lab Manual
II Year II Semester (R20)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision of the Department

The Computer Science & Engineering aims at providing continuously stimulating educational environment to its students for attaining their professional goals and meet the global challenges.

Mission of the Department

- **DM1:** To develop a strong theoretical and practical background across the computer science discipline with an emphasis on problem solving.
- **DM2:** To inculcate professional behaviour with strong ethical values, leadership qualities, innovative thinking and analytical abilities into the student.
- **DM3:** Expose the students to cutting edge technologies which enhance their employability and knowledge.
- **DM4:** Facilitate the faculty to keep track of latest developments in their research areas and encourage the faculty to foster the healthy interaction with industry.

Program Educational Objectives (PEOs)

- **PEO1:** Pursue higher education, entrepreneurship and research to compete at global level.
- **PEO2:** Design and develop products innovatively in computer science and engineering and in other allied fields.
- **PEO3:** Function effectively as individuals and as members of a team in the conduct of interdisciplinary projects; and even at all the levels with ethics and necessary attitude.
- **PEO4:** Serve ever-changing needs of society with a pragmatic perception.

PROGRAMME OUTCOMES (POs):

PO 1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO 2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO 3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO 4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO 5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the

	knowledge of, and need for sustainable development.
PO 8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO 11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAMME SPECIFIC OUTCOMES (PSOs):

PSO 1	The ability to apply Software Engineering practices and strategies in software project development using open-source programming environment for the success of organization.
PSO 2	The ability to design and develop computer programs in networking, web applications andIoT as per the society needs.
PSO 3	To inculcate an ability to analyze, design and implement database applications.

AIM: Demonstrating the following data preprocessing tasks using python libraries

Description/Theory:

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

PROCEDURE/CODE:

a) Loading the dataset

```
import numpy as np
import pandas as pd
dataset = pd.read_csv('D:/DM/file1.csv')
print(dataset)
x=dataset.iloc[:,:-1].values
```

```
print(x)
```

b) Identifying the dependent and independent variables.

```
dataset = pd.read_csv('D:/DM/file2.csv')
print(dataset)
x=dataset.iloc[:,:-1].values
y=dataset.iloc[:,-1].values
print(x)
print(y)
x=dataset.iloc[1:,1:-1].values
y=dataset.iloc[:,[0,4]].values
z=dataset.iloc[:0].values
print(x)
print(y)
print(z)
```

c) Dealing with missing data

```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer=imputer.fit(x[:,1:3])
x[:,1:3]=imputer.transform(x[:,1:3])
```

```
print(x)2
imputer=imputer.fit(x[:,1:])
x[:,1:]=imputer.transform(x[:,1:])
print(x)


import numpy as np
from sklearn.impute import SimpleImputer
imp=SimpleImputer(missing_values=np.nan,strategy='mean')
imp.fit([[1,2],[np.nan,3],[7,6]])
SimpleImputer()
x=[[np.nan,2],[6,np.nan],[7,6]]
print(imp.transform(x))
```

OUTPUT:

a)

```
Roll    Name Age Branch  Marks1  Marks2  Marks3
0    1  Ramesh    19    CSE      28      27      25
1     2  Suresh   19     IT      24      29      26
2     3  Geetha   18    EEE      19      28      27
3     4  Seetha   19    ECE      26      19      28
4     5  Sheela   19   MECH      27      18      29
5     6    Ram    20    CSE      20      27      20
6     7    Ravi   19     IT      21      26      30
7     8    Mani   19    ECE      30      25      19
8     9   Srinu   20    EEE      28      24      18
9    10    Devi   19    CSE      18      23      23
[[1 'Ramesh' 19 'CSE' 28 27]
 [2 'Suresh' 19 'IT' 24 29]
 [3 'Geetha' 18 'EEE' 19 28]
 [4 'Seetha' 19 'ECE' 26 19]
 [5 'Sheela' 19 'MECH' 27 18]
 [6 'Ram ' 20 'CSE' 20 27]
 [7 'Ravi' 19 'IT' 21 26]
 [8 'Mani' 19 'ECE' 30 25]
 [9 'Srinu' 20 'EEE' 28 24]
 [10 'Devi' 19 'CSE' 18 23]]
```

b)

```
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 69000.0 Yes
[['France' 'Male' 44.0 72000.0]
['Spain' 'Female' 27.0 48000.0]
['Germany' 'Male' 30.0 54000.0]

c)[['Female' 27.0 48000.0]
['Male' 30.0 54000.0]
['Male' 38.0 61000.0]
['Female' 40.0 63000.0]
['Female' 35.0 58000.0]
['Female' 38.125 52000.0]
['Male' 48.0 79000.0]
['Male' 50.0 83000.0]
['Male' 37.0 69000.0]]

[[4. 2.]
[6. 3.66666667]
[7. 6.]]

AIM: Demonstrate the following data preprocessing tasks using python libraries

Description/Theory:

Categorical data is a type of data that is used to group information with similar characteristics, while numerical data is a type of data that expresses information in the form of numbers.

Example of categorical data: gender

Categorical variables can be divided into two categories**:**

Nominal: no particular order

Ordinal: there is some order between values

PROCEDURE/CODE:

a) Dealing with categorical data.

```
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[0])],remainder='passthroug
h')
x=np.array(ct.fit_transform(x))
print(x)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y=le.fit_transform(y)
print(y)
```

b) Scaling the features.

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train[:, 6:]=sc.fit_transform(x_train[:, 6:])
x_test[:, 6:]=sc.transform(x_test[:, 6:])
print(x_train)
print(x_test)
```

c) Splitting dataset into Training and Testing Sets

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=1)
print(x_train)
print(x_test)
print(y_train)
print(y_test)
```

OUTPUT

a)
```
[[0.0 1.0 0.0 0.0 'Male' 44.0 72000.0]
 [1.0 0.0 0.0 1.0 'Female' 27.0 48000.0]
 [1.0 0.0 1.0 0.0 'Male' 30.0 54000.0]
 [1.0 0.0 0.0 1.0 'Male' 38.0 61000.0]
 [1.0 0.0 1.0 0.0 'Female' 40.0 nan]
 [0.0 1.0 0.0 0.0 'Female' 35.0 58000.0]
 [1.0 0.0 0.0 1.0 'Female' nan 52000.0]
 [0.0 1.0 0.0 0.0 'Male' 48.0 79000.0]
 [1.0 0.0 1.0 0.0 'Male' 50.0 83000.0]
 [0.0 1.0 0.0 0.0 'Male' 37.0 69000.0]]

[0 1 0 0 1 1 0 1 0 1]
```

b)
```
[[1.0 0.0 0.0 1.0 'Female' nan -1.0182239953527132]
 [1.0 0.0 1.0 0.0 'Female' 40.0 nan]
 [0.0 1.0 0.0 0.0 'Male' 44.0 0.5834766714942513]
 [1.0 0.0 0.0 1.0 'Male' 38.0 -0.2974586952715791]
 [1.0 0.0 0.0 1.0 'Female' 27.0 -1.3385641287221062]
 [0.0 1.0 0.0 0.0 'Male' 48.0 1.1440719048906889]
 [1.0 0.0 1.0 0.0 'Male' 50.0 1.4644120382600818]
 [0.0 1.0 0.0 0.0 'Female' 35.0 -0.5377137952986238]]
[[1.0 0.0 1.0 0.0 'Male' 30.0 -0.8580539286680167]
 [0.0 1.0 0.0 0.0 'Male' 37.0 0.3432215714672067]]
```

c)
```
[[1.0 0.0 0.0 1.0 'Female' nan 52000.0]
 [1.0 0.0 1.0 0.0 'Female' 40.0 nan]
 [0.0 1.0 0.0 0.0 'Male' 44.0 72000.0]
 [1.0 0.0 0.0 1.0 'Male' 38.0 61000.0]
 [1.0 0.0 0.0 1.0 'Female' 27.0 48000.0]
 [0.0 1.0 0.0 0.0 'Male' 48.0 79000.0]
 [1.0 0.0 1.0 0.0 'Male' 50.0 83000.0]
 [0.0 1.0 0.0 0.0 'Female' 35.0 58000.0]]
[[1.0 0.0 1.0 0.0 'Male' 30.0 54000.0]
 [0.0 1.0 0.0 0.0 'Male' 37.0 69000.0]]
[0 1 0 0 1 1 0 1]
[0 1]
```

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:

The Pearson correlation coefficient, often referred to as Pearson's r, is a measure of linear correlation between two variables. This means that the Pearson correlation coefficient measures a normalized measurement of covariance (i.e., a value between -1 and 1 that shows how much variables vary together).

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 differen

The Jaccard similarity measures the similarity between two sets of data to see which members are shared and distinct. The Jaccard similarity is calculated by dividing the number of observations in both sets by the number of observations in either set ces in their magnitude or scale.

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.

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 = $|x1 – x2| + |y1 – y2|$

PROCEDURE/CODE:

a) Pearson's Correlation

```
#Pearson's Correlation
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

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

#define two lists or array
a=np.array([2,1,2,3,2,9])
b=np.array([3,4,2,4,5,5])
print("a : ",a)
print("b : ",b)

#Compute cosine Similarity
cosine=np.dot(a,b)/(norm(a)*norm(b))
print("Cosine Similarity : ",cosine)
```

c)Jaccard Similarity

```
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])
d=jaccard(a,b)
print("Distance : ",d)
```

d)Euclidean Distance

```
from sklearn.metrics.pairwise import euclidean_distances
X=[[0,1],[1,1]]
euclidean_distances(X,X)
#Calculating euclidean distance between 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)
```

minkowski Distance

```
from scipy.spatial import minkowski_distance
```

```
row1=[10,20,15,10,5]
row2=[12,24,18,8,7]
#Calculate distance (p=1)
dist=minkowski_distance(row1,row2,1)
print(dist)
#Calculate distance (p=2)
dist=minkowski_distance(row1,row2,2)
print(dist)
```

#pearson without using methods

```
import math
n=int(input('Enter number of rows : '))
x,y = [],[]
for i in range(n):
    print('Enter '+str(i+1)+ ' row x value : ',end=" ")
    x.append(int(input()))
    print('Enter '+str(i+1)+ ' row y value : ',end=" ")
    y.append(int(input()))
print("Entered values : ")
print(x)
print(y)
x2=[]     #List for storing x^
y2=[]     #List for storing y^2
xy=[]     #List for storing xy
for i in range(n):
    x2.append(math.pow(x[i],2))
    y2.append(math.pow(y[i],2))
    xy.append(x[i]*y[i])
print('x square : ',x2)
print('y square : ',y2)
print('xy value : ',xy)
rxy=sum(xy) // n-((sum(x)//n) * (sum(y) // n))    #rho xy
rx =math.sqrt(sum(x2) // n-math.pow((sum(x)//n),2))
ry =math.sqrt(sum(y2) // n-math.pow((sum(y)//n),2))
result = rxy//(rx*ry)
print("Pearson's correlation coefficient : ",result)
```

OUTPUT:

a)Pearson Correlation : PearsonRResult(statistic=-0.7000000000000001, pvalue=0.18812040437418737)

b) a : [2 1 2 3 2 9]
b : [3 4 2 4 5 5]

Cosine Similarity : 0.8188504723485274
c) Distance : 0.4

d)array([[0., 1.],
[1., 0.]])
6.082762530298219

e) 10

Minkowski distance
13.0
6.082762530298219

Pearson
Enter number of rows : 5
Enter 1 row x value : -2
Enter 1 row y value : 4
Enter 2 row x value : -1
Enter 2 row y value : 1
Enter 3 row x value : 0
Enter 3 row y value : 3
Enter 4 row x value : 1
Enter 4 row y value : 2
Enter 5 row x value : 2
Enter 5 row y value : 0
Entered values :
[-2, -1, 0, 1, 2]
[4, 1, 3, 2, 0]
x square : [4.0, 1.0, 0.0, 1.0, 4.0]
y square : [16.0, 1.0, 9.0, 4.0, 0.0]
xy value : [-8, -1, 0, 2, 0]
Pearson's correlation coefficient : -1.0

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

Description/Theory:

PROCEDURE/CODE:

Linear regression is the type of regression that forms a relationship between the target variable and one or more independent variables utilizing a straight line. The given equation represents the equation of linear regression

Y = a + b*X + e.

#Building a linear regression model for diabetes dataset

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets,linear_model
from sklearn.metrics import mean_squared_error,r2_score
diabetes_X,diabetes_Y = datasets.load_diabetes(return_X_y = True)
diabetes_X = diabetes_X[:,np.newaxis,2]

#Split data into training/testing sets
diabetes_X_trian = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

#Split the dependent variables
diabetes_y_train = diabetes_Y[:-20]
diabetes_y_test = diabetes_Y[-20:]

#Create an object for Linear Regression
regr = linear_model.LinearRegression()

#Train the model using the training sets
regr.fit(diabetes_X_trian,diabetes_y_train)

#Make predictions using training sets
diabetes_y_pred = regr.predict(diabetes_X_test)

print(diabetes_y_pred)
```

Plotting :

```
plt.scatter(diabetes_X_test,diabetes_y_test,color='black')
```

```
plt.plot(diabetes_X_test,diabetes_y_pred,color='blue',linewidth=2)
plt.show()
```

Grid:

```
plt.scatter(diabetes_X_test,diabetes_y_test,color='black')
plt.plot(diabetes_X_test,diabetes_y_pred,color='blue',linewidth=2,marker='*',markerfacecolor='
red')
plt.grid()
plt.show()
```

Linear Regression:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
x=np.array([5,15,45,25,35,45]).reshape((-1,1))
y=np.array([5,21,14,22,32,28])
print(x,y)

x=np.array([5,15,45,25,35,45]).reshape((-1,1))
y=np.array([5,21,14,22,32,28])
print(x,y)
model = LinearRegression()
model.fit(x,y)
result=model.score(x,y)

print("Score :  ",result)
model.fit(x,y)
result=model.score(x,y)
print("Score : ",result)
print("Intercept : ",model.intercept_)
print("Slope : ",model.coef_)
y_pred = model.predict(x)   #predicted by y value
print("Actual values of y : ",y);
print("Predicted values of y : ",y_pred)

plt.show()
plt.plot(x,y_pred,color='blue',linewidth=2,marker='o',markerfacecolor='red')

[<matplotlib.lines.Line2D at 0x23b679e6850>]

plt.plot(x,y_pred,color='blue',linewidth=2,marker='o',markerfacecolor='red')
plt.grid()
plt.show()
```

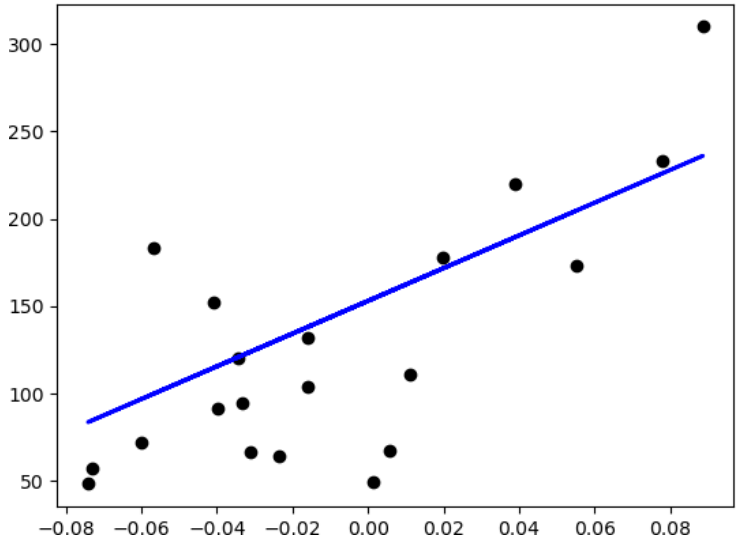
Multiple Linear Regression

```
#Importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#Importing the dataset
dataset=pd.read_csv('D:/DM/dataset.csv')
dataset.describe()
shape=dataset.shape
print(shape)
dataset.columns
X=dataset.iloc[:, :-1].values
y=dataset.iloc[:, -1].values
print(X)

#Encoding Categorical data
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[3])],remainder='passthroug
h')
X=np.array(ct.fit_transform(X))
print(X)

#Spliting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=0)

#Training the Multiple Linear Regression model on the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train)

#Predicting the Test set results
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1),y_test.reshape(len(y_test),1)),1))

plt.scatter(y_test,y_pred);
plt.xlabel('Actual')
plt.ylabel('Predicted')
pred_df=pd.DataFrame({'Actual Value' : y_test,'Predicted Value' :y_pred,'Difference' :y_test-
y_pred})
pred_df
```

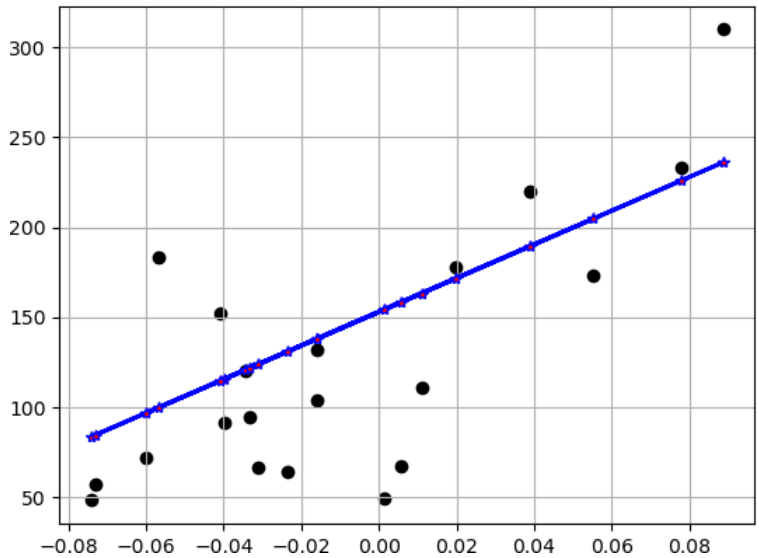
OUTPUT:

[225.9732401 115.74763374 163.27610621 114.73638965 120.80385422
 158.21988574 236.08568105 121.81509832 99.56772822 123.83758651
 204.73711411 96.53399594 154.17490936 130.91629517 83.3878227
 171.36605897 137.99500384 137.99500384 189.56845268 84.3990668]

Plotting:



GRID:

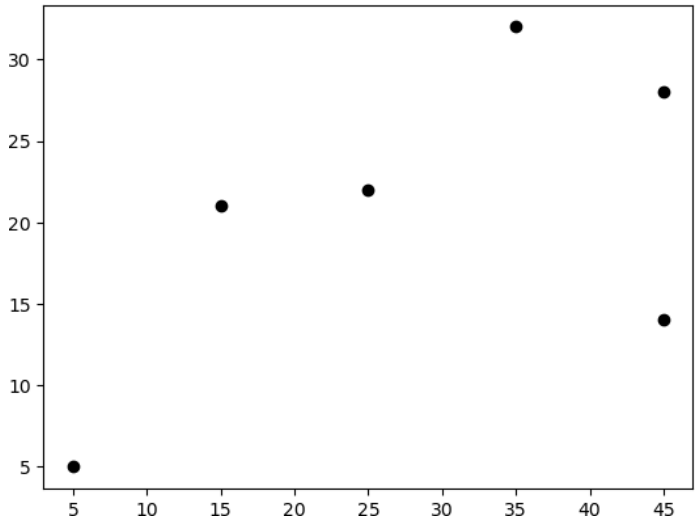


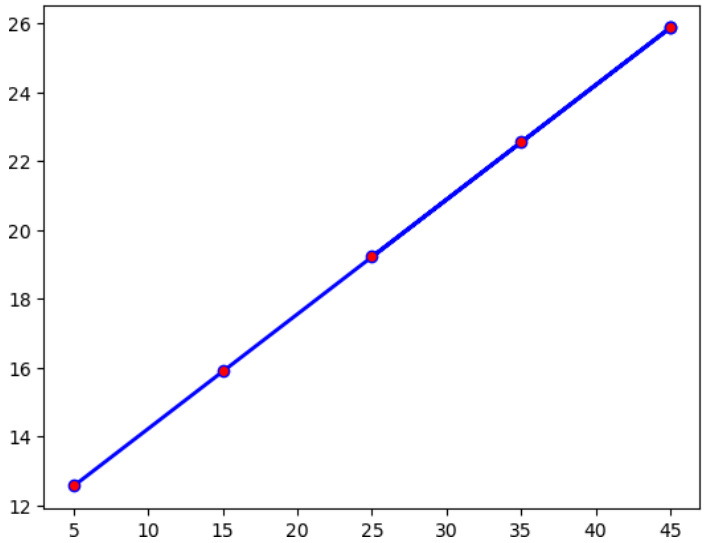
[[5]
 [15]
 [45]
 [25]
 [35]
 [45]] [5 21 14 22 32 28]

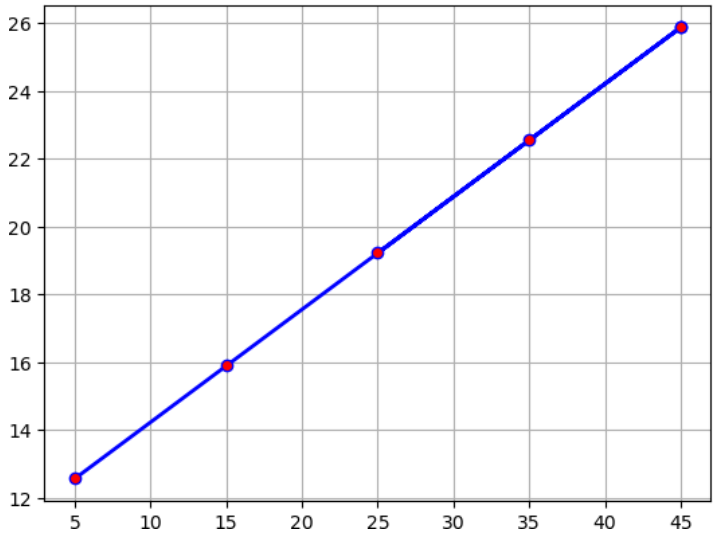
[[5]
 [15]

[45]
[25]
[35]
[45]] [5 21 14 22 32 28]
Score : 0.3114260563380282
Score : 0.3114260563380282
Intercept : 10.912499999999996
Slope : [0.3325]
Actual values of y : [5 21 14 22 32 28]
Predicted values of y : [12.575 15.9 25.875 19.225 22.55 25.875]

plt.scatter(x,y,color='black')





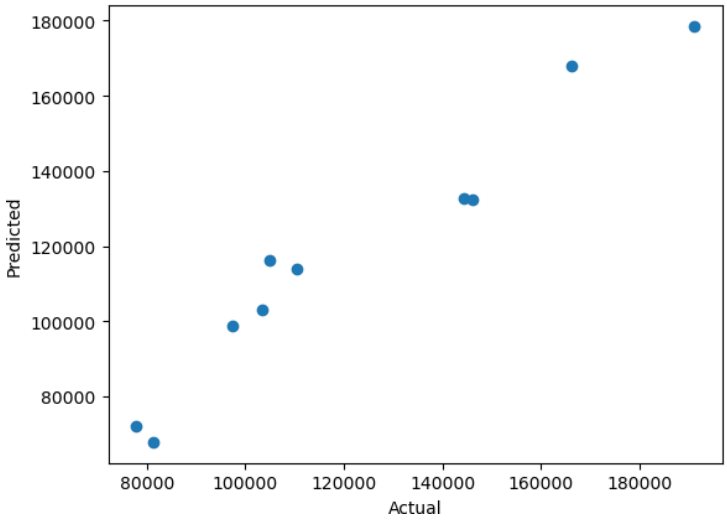


Multiple Linear Regression:

(50, 5)

[[165349.2 136897.8 471784.1 'New York']
[162597.7 151377.59 443898.53 'California']
[153441.51 101145.55 407934.54 'Florida']
[144372.41 118671.85 383199.62 'New York']
[142107.34 91391.77 366168.42 'Florida']
[131876.9 99814.71 362861.36 'New York']
[134615.46 147198.87 127716.82 'California']
[130298.13 145530.06 323876.68 'Florida']
[120542.52 148718.95 311613.29 'New York']
[123334.88 108679.17 304981.62 'California']
[101913.08 110594.11 229160.95 'Florida']
[100671.96 91790.61 249744.55 'California']
[93863.75 127320.38 249839.44 'Florida']
[91992.39 135495.07 252664.93 'California']
[119943.24 156547.42 256512.92 'Florida']

Actual Value	Predicted Value	Difference
103282.38	103015.201598	267.178402
144259.40	132582.277608	11677.122392
146121.95	132447.738452	13674.211548


180000
160000
140000
120000
100000
80000
Predicted
80000
100000
120000
140000
160000
180000
Actual

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

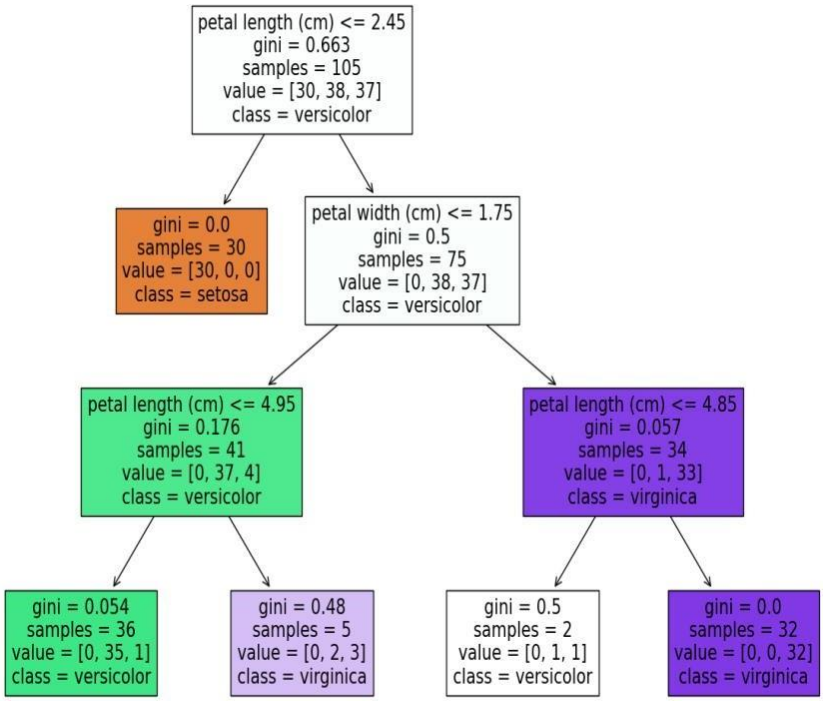
Description/Theory:
Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches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

PROCEDURE/CODE:

```
#import libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
#Load iris dataset
iris=load_iris()
X=iris.data
y=iris.target
#Split the data into training and testing sets
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=None)
#Create an instance of the DecisionTreeClassifier class
tree_clf=DecisionTreeClassifier(max_depth=3)
#Fit the model on the training data
tree_clf.fit(X_train,y_train)
DecisionTreeClassifier(max_depth=3)
#predict on the testing data
y_pred=tree_clf.predict(X_test)
#Calculate accuracy of the model
accuracy=accuracy_score(y_test,y_pred)
print('Accuracy : ',accuracy)
#Visualize the decision tree using the plot_tree function
plt.figure(figsize=(15,10))
plot_tree(tree_clf,filled=True,feature_names=iris.feature_names,class_names=iris.target_names
)
plt.show()
```

OUTPUT:

Accuracy : 0.9555555555555556



AIM: Apply Naive 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.
Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles**.**

PROCEDURE/CODE:

```
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
dataset=pd.read_csv('D:\DM\iris.csv')
print(dataset)

X=dataset.iloc[:,:4].values
Y=dataset['Species'].values
print(Y)
print(X)

#split the dataset into training and test datasets
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3)
#create an object for Bayes Classifier GaussianNB
classifier=GaussianNB()
classifier.fit(X_train,Y_train)
#predict the values
print(X_test[0])
y_pred=classifier.predict(X_test)
print(y_pred)
accuracy=accuracy_score(Y_test,y_pred)
print("Accuracy: ",accuracy)
```

```
#build confusion matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(Y_test,y_pred)
from sklearn.metrics import accuracy_score
print("Accuracy: ",accuracy_score(Y_test,y_pred))
cm

df=pd.DataFrame({'Real Values' : Y_test, 'Predicted Values' : y_pred})
print(df)
```

Naive Bayes :

```
#predict class table for
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import numpy as np
#import iris dataset
iris=load_iris()
X=iris.data
Y=iris.target
le=LabelEncoder()
Y=le.fit_transform(Y)
#Split the dataset into training and testing sets
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3, random_state=42)
#Train a Naive bayes model on the training data
nb_model = GaussianNB()
nb_model.fit(X_train, Y_train)
#Make predictions on the test data
y_pred = nb_model.predict(X_test)
y_pred = le.inverse_transform(y_pred)
accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy: ",accuracy)
new_observation = np.array([[5.8, 3.0, 4.5, 1.5]])
predicted_class = nb_model.predict(new_observation)
predicted_class = le.inverse_transform(predicted_class)
print("Predicted class: ", predicted_class)
```

OUTPUT:

```
 Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0    1            5.1           3.5            1.4           0.2
1    2            4.9           3.0            1.4           0.2
```

```
2    3          4.7          3.2          1.3          0.2

3    4          4.6          3.1          1.5          0.2
4    5          5.0          3.6          1.4          0.2
..  ...         ...          ...          ...          ...
145  146          6.7          3.0          5.2          2.
146  147          6.3          2.5          5.0          1.9
147  148          6.5          3.0          5.2          2.0
148  149          6.2          3.4          5.4          2.3
149  150          5.9          3.0          5.1          1.8

           Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa']
[[ 1.   5.1  3.5  1.4]
 [ 2.   4.9  3.   1.4]
 [ 3.   4.7  3.2  1.3]
 [ 4.   4.6  3.1  1.5]
 [ 5.   5.   3.6  1.4]
 [ 6.   5.4  3.9  1.7]
 [ 7.   4.6  3.4  1.4]
 [ 8.   5.   3.4  1.5]
 [ 9.   4.4  2.9  1.4]
 [ 10.  4.9  3.1  1.5]
 [ 11.  5.4  3.7  1.5]
['Iris-setosa' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-setosa'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica'
 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'
```

Accuracy: 1.0
Accuracy: 1.0

	Real Values	Predicted Values
0	Iris-setosa	Iris-setosa
1	Iris-virginica	Iris-virginica
2	Iris-versicolor	Iris-versicolor
3	Iris-setosa	Iris-setosa
4	Iris-versicolor	Iris-versicolor
5	Iris-virginica	Iris-virginica

Accuracy: 0.9777777777777777
Predicted class: [1]

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

Description/Theory:
Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

PROCEDURE/CODE:

```
pip install mlxtend

import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori,association_rules
data =[['Bread', 'Milk', 'Eggs'],
      ['Bread', 'Diapers', 'Beer', 'Eggs'],
      ['Milk', 'Diapers', 'Beer', 'Cola'],
      ['Bread' 'Milk', 'Diapers', 'Beer', 'Cola', 'Eggs']]
te=TransactionEncoder()
te_ary = te.fit(data).transform(data)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
frequent_itemsets = apriori(df, min_support=0.75, use_colnames=True)
print(frequent_itemsets)

rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=0.7)
print(rules)
selected_columns = ['antecedents', 'consequents', 'antecedent support', 'consequent support',
'support', 'confidence']
print(rules[selected_columns])
```

Apriori:

```
pip install apyori
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
from apyori import apriori
store_data = pd.read_csv("D:/DM/store_data.csv",header=None)
display(store_data.head())
store_data.shape

records = []
for i in range(1,7501):
    records.append([str(store_data.values[i,j]) for j in range(0,20)])
print(type(records))

association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3,
min_length=2)
association_results = list(association_rules)
print("There are {} Relation derived.".format (len(association_results)))
for i in range(0, len(association_results)):
    print(association_results[i][0])

for item in association_results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule: "+ items[0] + "->" +items[1])
    print("Support: " +str(item[1]))
    print("Confidence: " +str(item[2][0][2]))
    print("Lift: " +str(item[2][0][3]))
    print("=========================================")
```

OUTPUT:

```
  support        itemsets
0    0.75          (Beer)
1    0.75       (Diapers)
2    0.75          (Eggs)
3    0.75  (Beer, Diapers)
antecedents consequents antecedent support  consequent support  support  \
0     (Beer)   (Diapers)              0.75              0.75    0.75
1   (Diapers)     (Beer)              0.75              0.75    0.75

   confidence      lift leverage  conviction  zhangs_metric
0         1.0 1.333333   0.1875         inf            1.0
1         1.0 1.333333   0.1875         inf            1.0
  antecedents consequents antecedent support  consequent support  support  \
0     (Beer)   (Diapers)              0.75              0.75    0.75
1   (Diapers)     (Beer)              0.75              0.75    0.75
```

confidence
0 1.0
1 1.0
2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole wheat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	frozen smoothie	spinach	olive oil
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

(7501, 20)

<class 'list'>
There are 48 Relation derived.
frozenset({'chicken', 'light cream'})
frozenset({'escalope', 'mushroom cream sauce'})
frozenset({'escalope', 'pasta'})
frozenset({'ground beef', 'herb & pepper'})

frozenset({'tomato sauce', 'ground beef'})
frozenset({'whole wheat pasta', 'olive oil'})
frozenset({'shrimp', 'pasta'})
frozenset({'nan', 'chicken', 'light cream'})
frozenset({'shrimp', 'frozen vegetables', 'chocolate'})
frozenset({'cooking oil', 'ground beef', 'spaghetti'})

Rule: chicken->light cream
Support: 0.0045333333333333334
Confidence: 0.2905982905982906
Lift: 4.843304843304844
==========
Rule: escalope->mushroom cream sauce
Support: 0.005733333333333333
Confidence: 0.30069930069930073
Lift: 3.7903273197390845

AIM: Apply K- Means clustering algorithm on any dataset

Description/Theory:
K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters.
The k-means clustering algorithm mainly performs two tasks:
Determines the best value for K center points or centroids by an iterative process.
Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.
Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

PROCEDURE/CODE:

```
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)

print("\nClusters:",kmeans.cluster_centers_)

print("\nLabels: ",kmeans.labels_)

plt.scatter(X[:,0],X[:,1],c=kmeans.labels_,cmap="rainbow")
```

OUTPUT:

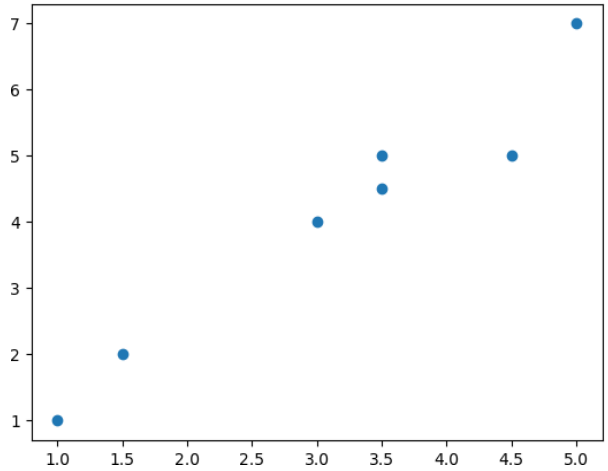
```
[[1.  1. ]

 [1.5 2. ]

 [3.  4. ]

 [5.  7. ]
```

[3.5 5.]

[4.5 5.]

[3.5 4.5]]

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

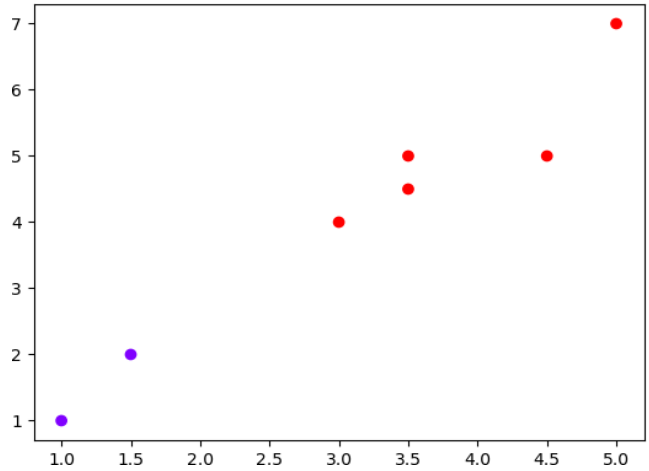


KMeans(n_clusters=2)

Clusters: [[1.25 1.5]

[3.9 5.1]]

Labels: [0 0 1 1 1 1 1]



AIM: Apply Hierarchical Clustering algorithm on any dataset

Description/Theory:

Hierarchical Clustering in Machine Learning

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.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

PROCEDURE/CODE:

```
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=np.random.random_sample(size=5)

b=np.random.random_sample(size=5)

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

plt.figure(figsize=(8,5))

plt.xlabel('Column a')

plt.ylabel('Column b')
```

```
plt.title('Scatter plot of x and y')
plt.scatter(data['a'],data['b'],c='r',marker='*')
for j in data.itertuples():plt.annotate
   (j.index,(j.a,j.b),fontsize=15)
dist=pd.DataFrame(squareform(pdist(data[['a','b']]),'euclidean'),
columns=data.index.values,
index=data.index.values)
dist
```

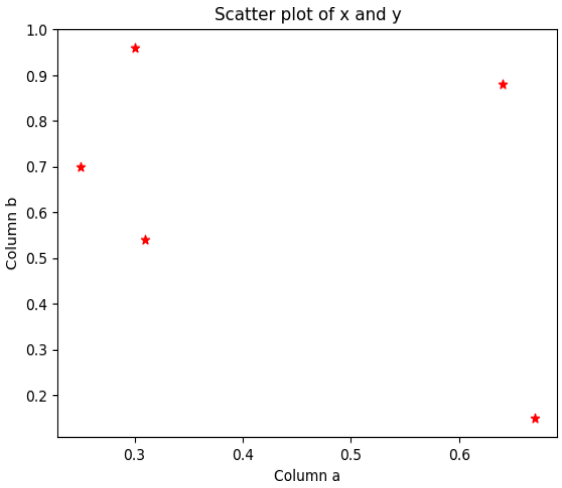
DANDROGRAM

```
plt.figure(figsize=(12,5))
plt.title('Dendrogram with Single linkage')
dend=shc.dendrogram(shc.linkage(data[['a','b']],method='single'),labels=data.index)
```

OUTPUT:

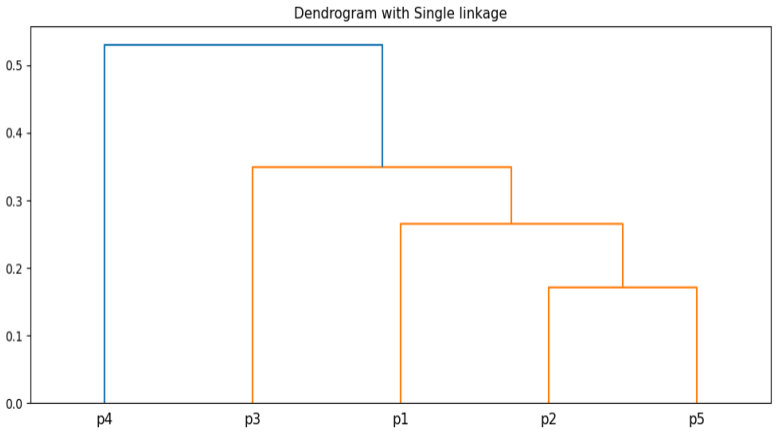
	a	b
Point		
p1	0.30	0.96
p2	0.31	0.54
p3	0.64	0.88
p4	0.67	0.15
p5	0.25	0.70

<Figure size 800x500 with 0 Axes>
<Figure size 800x500 with 0 Axes>



	p1	p2	p3	p4	p5
p1	0.000000	0.420119	0.349285	0.890505	0.264764
p2	0.420119	0.000000	0.473814	0.530754	0.170880
p3	0.349285	0.473814	0.000000	0.730616	0.429535
p4	0.890505	0.530754	0.730616	0.000000	0.692026
p5	0.264764	0.170880	0.429535	0.692026	0.000000

Dandrogram



AIM: Apply DBSCAN clustering algorithm on any dataset

Description/Theory:
Steps Used In DBSCAN Algorithm
Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.
For each core point if it is not already assigned to a cluster, create a new cluster.
Find recursively all its density-connected points and assign them to the same cluster as the core point.
A point a and b are said to be density connected if there exists a point c which has a sufficient number of points in its neighbors and both points a and b are within the eps distance. This is a chaining process. So, if b is a neighbor of c, c is a neighbor of d, and d is a neighbor of e, which in turn is neighbor of a implying that b is a neighbor of a.
Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

PROCEDURE/CODE:

```
import numpy as np

from sklearn.datasets import make_blobs

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

from sklearn.cluster import DBSCAN

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

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

print(X,y)

db=DBSCAN(eps=0.4,min_samples=5)

db.fit(X)

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)
```

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

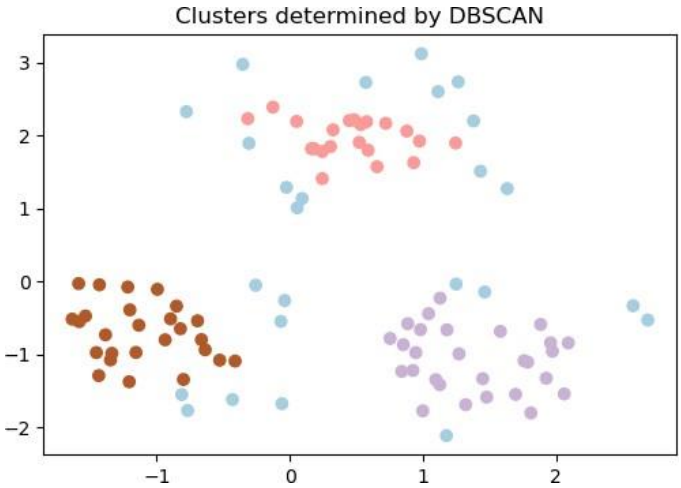
Kmeans for DBSCAN:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
plt.scatter(X[:,0],X[:,1])
kmeans=KMeans(n_clusters=2)
kmeans.fit(X)
print("\n Clusters:",kmeans.cluster_centers_)
print("\n Labels:",kmeans.labels_)
```

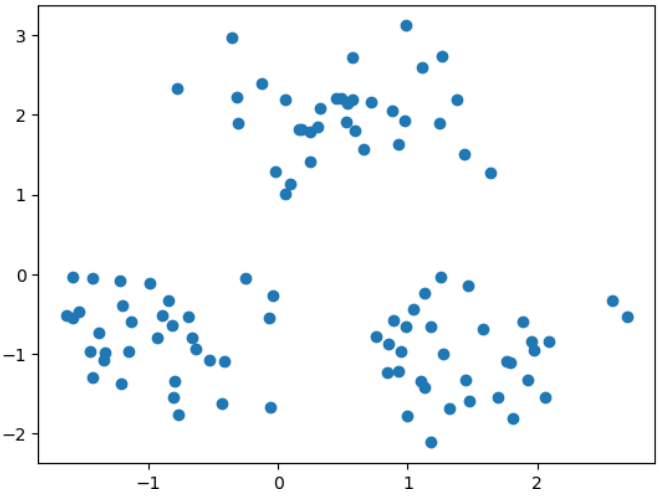
OUTPUT:

```
[[ 0.57747371 2.18908126]
 [ 1.1781908  -2.11170158]
 [ 0.53325861  2.15123595]
 [ 0.94780833 -0.97391746]
 [ 1.27223375 -0.99126042]
 [ 1.26638961  2.73467938]
 [ 0.58871307 1.79910953]
DBSCAN(eps=0.4)
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)
[ 0 -1  0  1  1 -1  0  1  2 -1  1  2  2  1 -1 -1 -1 2  1  2  2  0  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]



KMeans(n_clusters=2)



Clusters: [[0.25333922 -0.89314775]
[0.49210963 2.00254955]]

Labels: [1 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0
1 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1 0
1 1 1 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0]