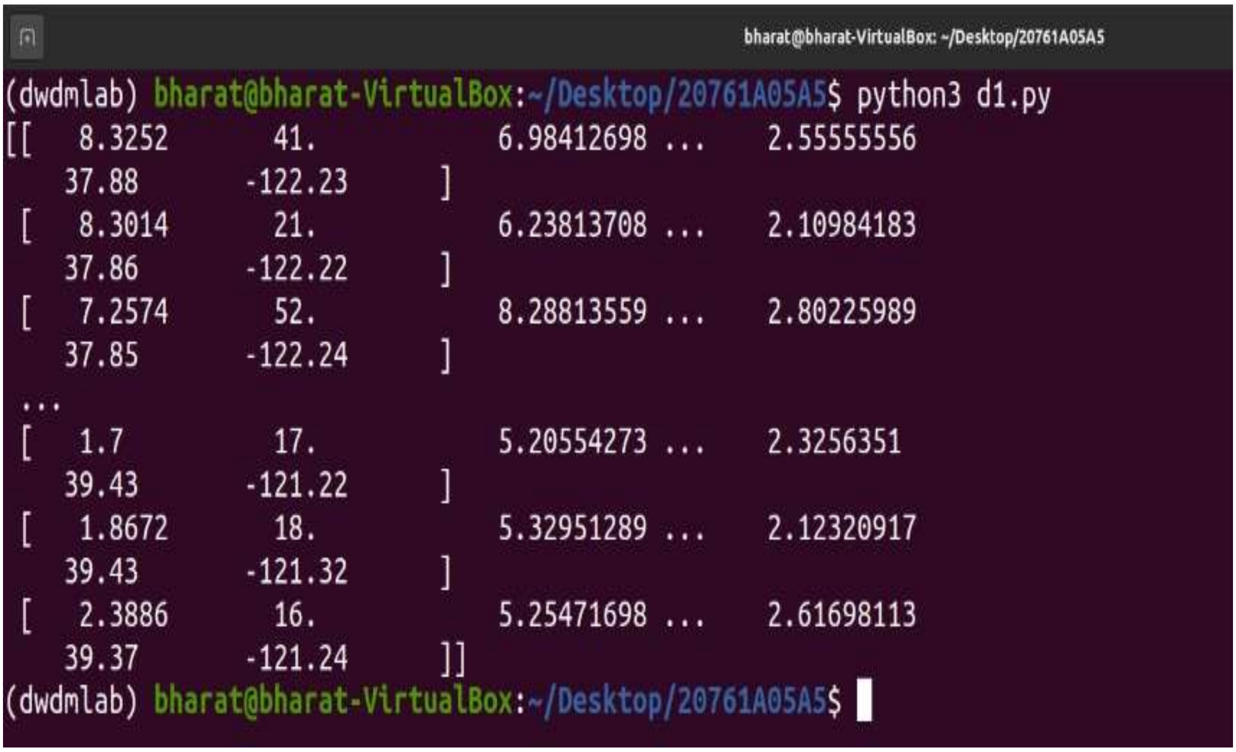
1. Demonstrate the following data preprocessing tasks using python libraries.

a) Loading the dataset

Program:

```
from sklearn import datasets

from sklearn.datasets import fetch_california_housing

df=fetch_california_housing()

x=df.data

print(x)
```

Output:



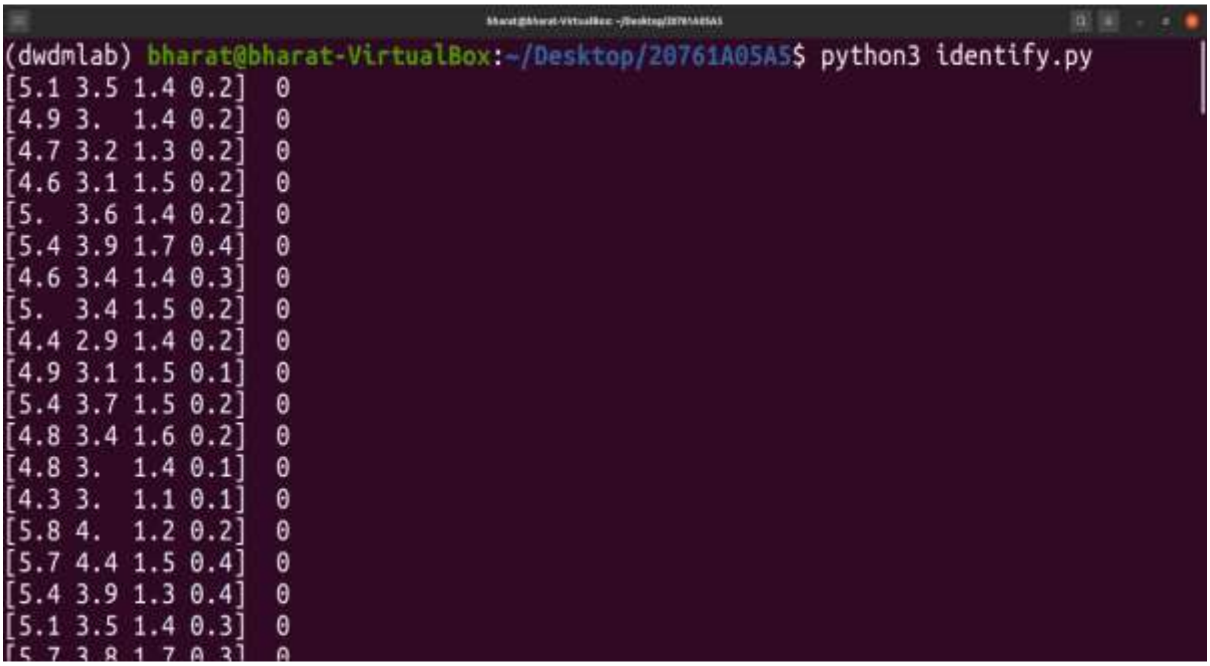
```
bharat@bharat-VirtualBox: ~/Desktop/20761A05A5
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 d1.py
[[   8.3252       41.            6.98412698 ...    2.55555556
    37.88       -122.23      ]
 [   8.3014       21.            6.23813708 ...    2.10984183
    37.86       -122.22      ]
 [   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      ]]
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$
```

b) Identifying the dependent and independent variables.

Programm:

```
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:



```
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 identify.py
[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
[5.7 3.8 1.7 0.3]  0
```

c) Dealing with missing data

Programm:

import numpy as np

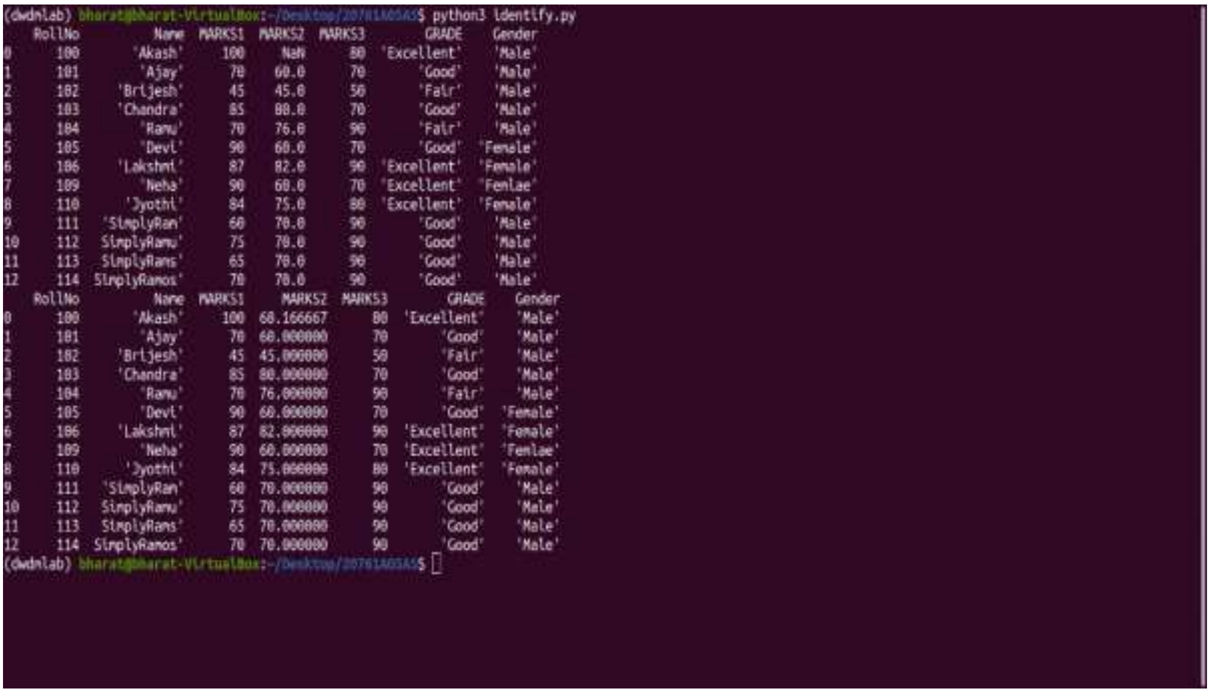
import pandas as pd

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

print(df)

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

print(df)



```
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/[illegible]$ python3 identify.py
    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'
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/[illegible]$
```

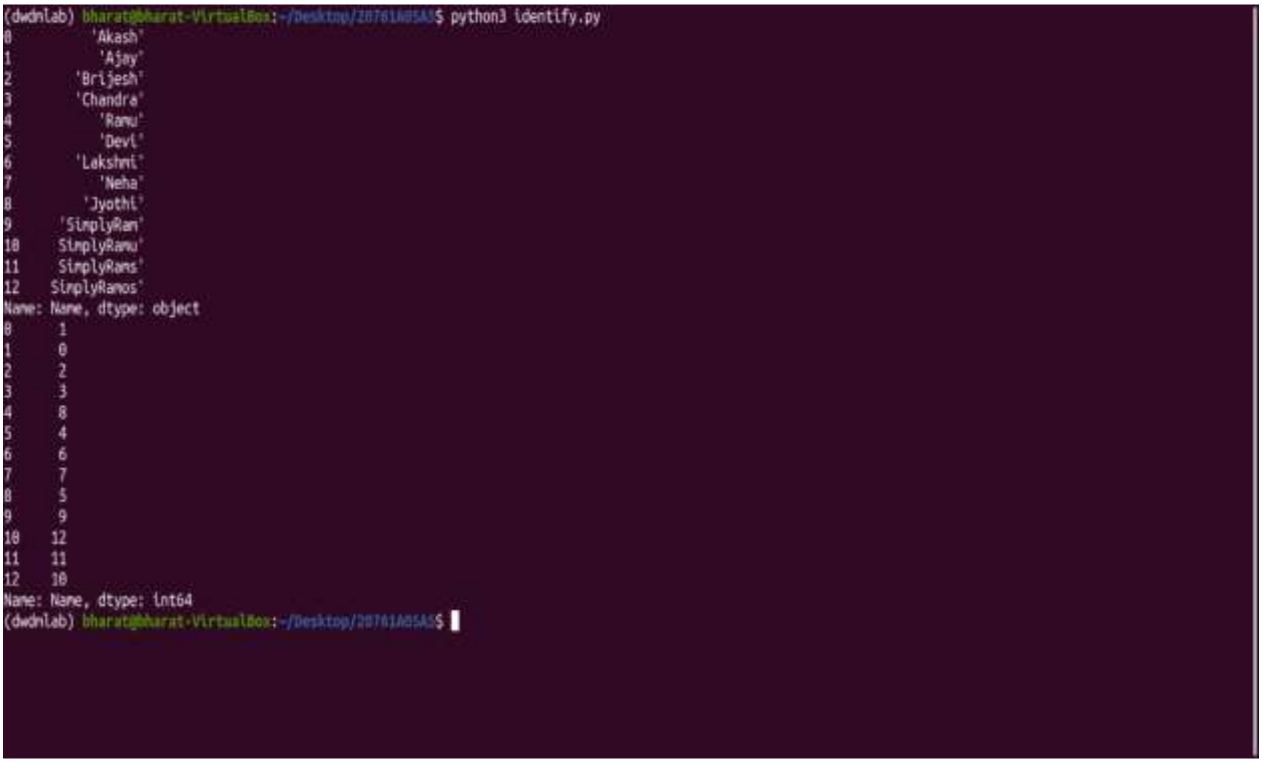
2. Demonstrate the following data preprocessing tasks using python libraries.

a) Dealing with categorical data

Programm: 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:



Programm: 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)
```

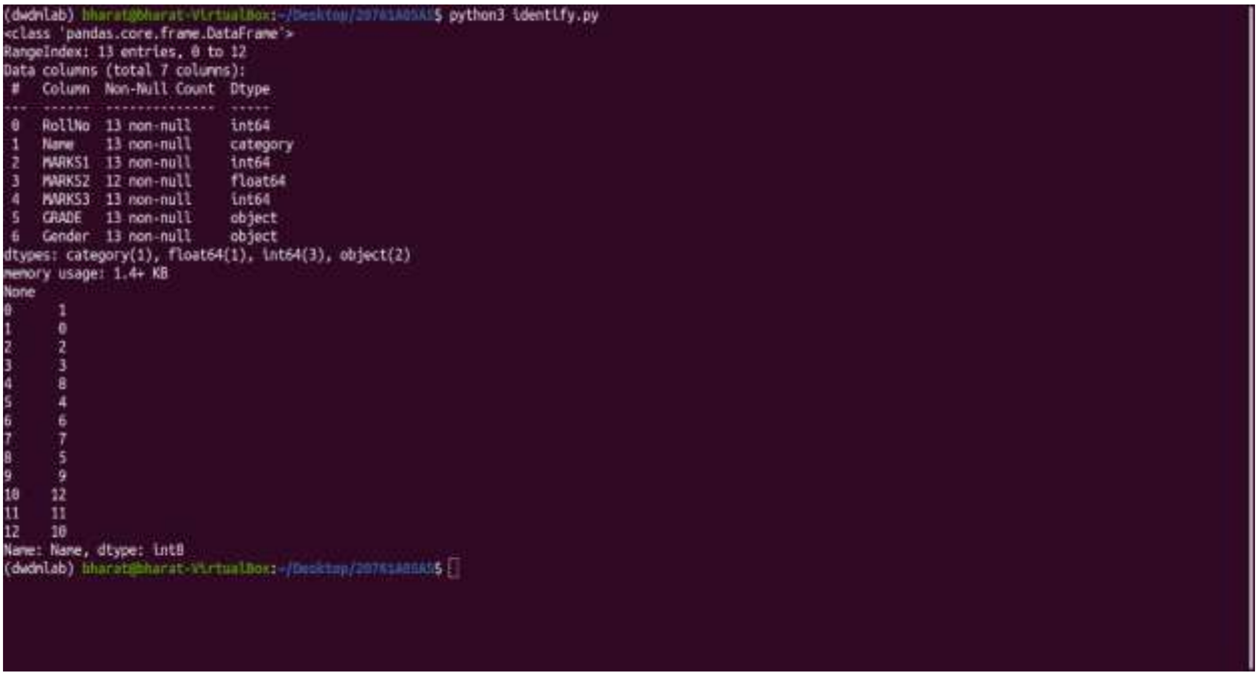
Output:

```
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 identify.py
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]]
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$
```

Programm: 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:



```
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 identify.py
<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
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$
```

b) Scaling the features

```
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:

```
(dwdmlab) bharat@bharat-VirtualBox:~/Desktop/20761A05A5$ python3 iri.py
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]
 [1.         0.08474576]
 [0.79166667 0.05084746]
 [0.625      0.06779661]
 [0.75       0.11864407]
 [0.75       0.08474576]
 [0.58333333 0.11864407]
 [0.70833333 0.08474576]
 [0.66666667 0.        ]
 [0.54166667 0.11864407]
 [0.58333333 0.15254237]
 [0.41666667 0.10169492]
 [0.58333333 0.10169492]
 [0.625      0.08474576]
```

```
 [0.58333333 0.74576271]
 [0.41666667 0.69491525]]
Standard Scaler output is
 [[ 1.03205722 -1.3412724 ]
 [-0.1249576  -1.3412724 ]
 [ 0.33784833 -1.39813811]
 [ 0.10644536 -1.2844067 ]
 [ 1.26346019 -1.3412724 ]
 [ 1.95766909 -1.17067529]
 [ 0.80065426 -1.3412724 ]
 [ 0.80065426 -1.2844067 ]
 [-0.35636057 -1.3412724 ]
 [ 0.10644536 -1.2844067 ]
 [ 1.49486315 -1.2844067 ]
 [ 0.80065426 -1.227541  ]
 [-0.1249576  -1.3412724 ]
 [-0.1249576  -1.51186952]
 [ 2.18907205 -1.45500381]
 [ 3.11468391 -1.2844067 ]
 [ 1.95766909 -1.39813811]
 [ 1.03205722 -1.3412724 ]
 [ 1.72626612 -1.17067529]
 [ 1.72626612 -1.2844067 ]
 [ 0.80065426 -1.17067529]
 [ 1.49486315 -1.2844067 ]
```

c) Splitting dataset into Training and Testing Sets

Programm:

```
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:

3. Demonstrate the following Similarity and Dissimilarity Measures using python

a) Pearson’s Correlation

Programm:

```
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

Programm:

```
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

Programm:

```
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) Manhattan Distance

Programm:

```
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) Euclidean Distance

Programm:

from sklearn.metrics.pairwise import euclidean_distances

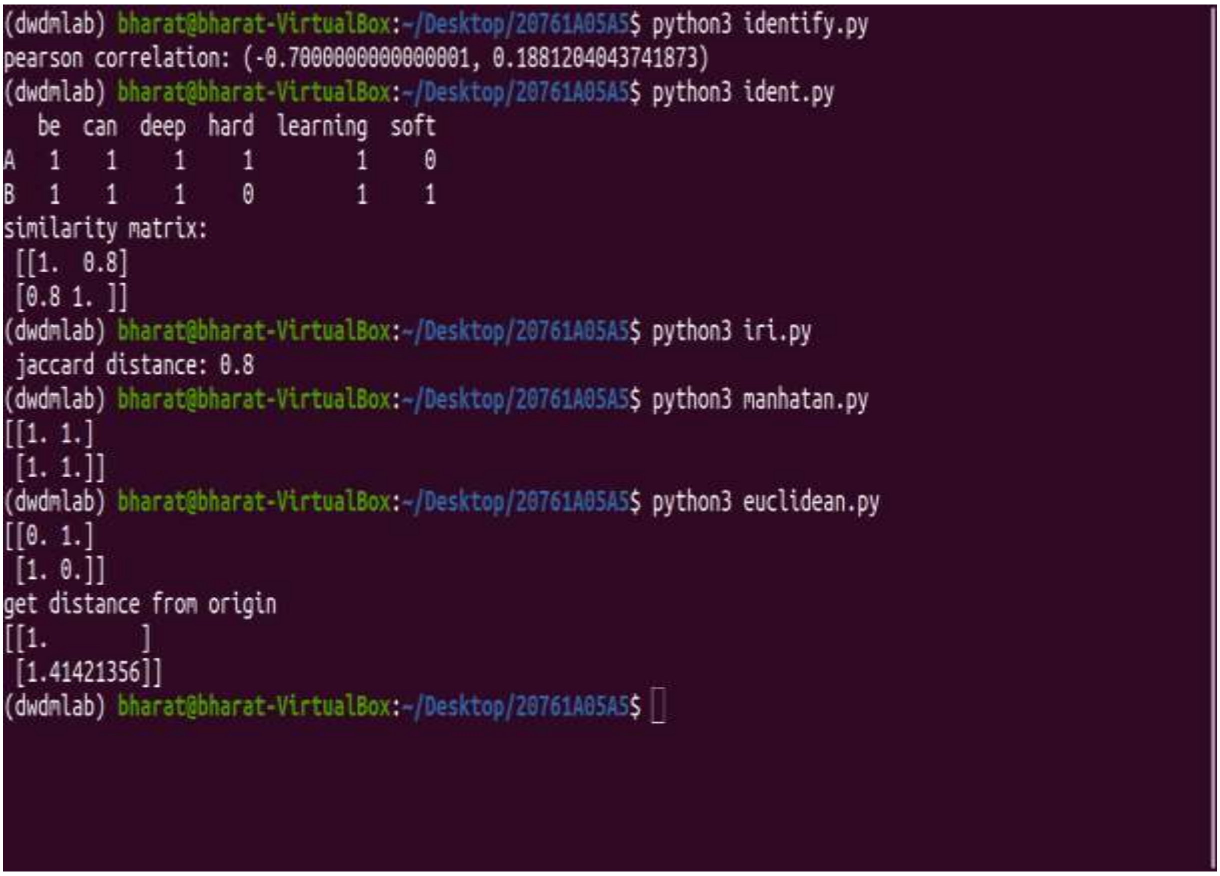
X=[[0,1],[1,1]]

print(euclidean_distances(X,X))

print("get distance from origin")

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

Output:



4. Build a model using linear regression algorithm on any dataset.

Programm:

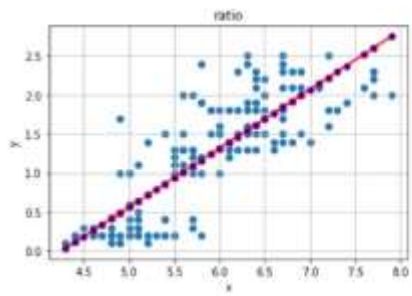
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error
import matplotlib.pyplot as plt
df=pd.read_csv('iris.csv')
X=np.array(df['sepal_length']).reshape((-1,1))
Y=df['petal_width']
m=LinearRegression()
print(m.fit(X,Y))
print("slope is:",m.coef_)
print("intercept is :",m.intercept_)
print("score is",m.score(X,Y))
Y_pred=m.predict(X)
print("PREDICTED VALUES OF Y",Y_pred)
print("r2 score values are:",r2_score(Y,Y_pred))
print("mean squared error values:",mean_squared_error(Y,Y_pred))
plt.scatter(X,Y)
plt.plot(X,Y_pred,c='red',marker='o',lw='1.5',markerfacecolor='blue')
plt.grid()
plt.xlabel("x")
plt.ylabel("y")
```

plt.title("ratio")

Output:

```
LinearRegression()
slope is: [0.75384088]
intercept is : -3.2062768959723345
score is 0.6690481463418159
PREDICTED VALUES OF Y [0.63831161 0.48754343 0.33677526 0.26139117 0.56292752 0.86446387
 0.26139117 0.56292752 0.11062299 0.48754343 0.86446387 0.41215934
 0.41215934 0.0352389  1.16600023 1.09061614 0.86446387 0.63831161
 1.09061614 0.63831161 0.86446387 0.63831161 0.26139117 0.63831161
 0.41215934 0.56292752 0.56292752 0.7136957  0.7136957  0.33677526
 0.41215934 0.86446387 0.7136957  0.93984796 0.48754343 0.56292752
 0.93984796 0.48754343 0.11062299 0.63831161 0.56292752 0.18600708
 0.11062299 0.56292752 0.63831161 0.41215934 0.63831161 0.26139117
 0.78907979 0.56292752 2.07060929 1.61830476 1.9952252  0.93984796
 1.69368885 1.09061614 1.54292067 0.48754343 1.76907294 0.7136957
 0.56292752 1.24138432 1.31676841 1.39215249 1.01523205 1.84445702
 1.01523205 1.16600023 1.46753658 1.01523205 1.24138432 1.39215249
 1.54292067 1.39215249 1.61830476 1.76907294 1.91984111 1.84445702
 1.31676841 1.09061614 0.93984796 0.93984796 1.16600023 1.31676841
 0.86446387 1.31676841 1.84445702 1.54292067 1.01523205 0.93984796
 0.93984796 1.39215249 1.16600023 0.56292752 1.01523205 1.09061614
 1.09061614 1.46753658 0.63831161 1.09061614 1.54292067 1.16600023
 2.14599338 1.54292067 1.69368885 2.52291382 0.48754343 2.29676155
 1.84445702 2.22137747 1.69368885 1.61830476 1.91984111 1.09061614
 1.16600023 1.61830476 1.69368885 2.59829791 2.59829791 1.31676841
 1.9952252  1.01523205 2.59829791 1.54292067 1.84445702 2.22137747
 1.46753658 1.39215249 1.61830476 2.22137747 2.37214564 2.74906608
 1.61830476 1.54292067 1.39215249 2.59829791 1.54292067 1.61830476
 1.31676841 1.9952252  1.84445702 1.9952252  1.16600023 1.91984111
 1.84445702 1.84445702 1.54292067 1.69368885 1.46753658 1.24138432]
r2 score values are: 0.6690481463418159
mean squared error values: 0.19146609071086385

Out[1]: Text(0.5, 1.0, 'ratio')
```


```
r2 score values are: 0.6690481463418159
mean squared error values: 0.19146609071086385

Out[1]: Text(0.5, 1.0, 'ratio')
```

Programm:By splitting the Data

import pandas as pd

import numpy as np

from sklearn.metrics import r2_score,mean_squared_error

import matplotlib.pyplot as plt

from sklearn import linear_model

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

X=np.array(df['sepal_length']).reshape((-1,1))

Y=df['petal_width']

X_train=X[:20]

X_test=X[-20:]

Y_train=Y[:20]

Y_test=Y[-20:]

r=LinearRegression()

r.fit(X_train,Y_train)

Y_pred=r.predict(X_test)

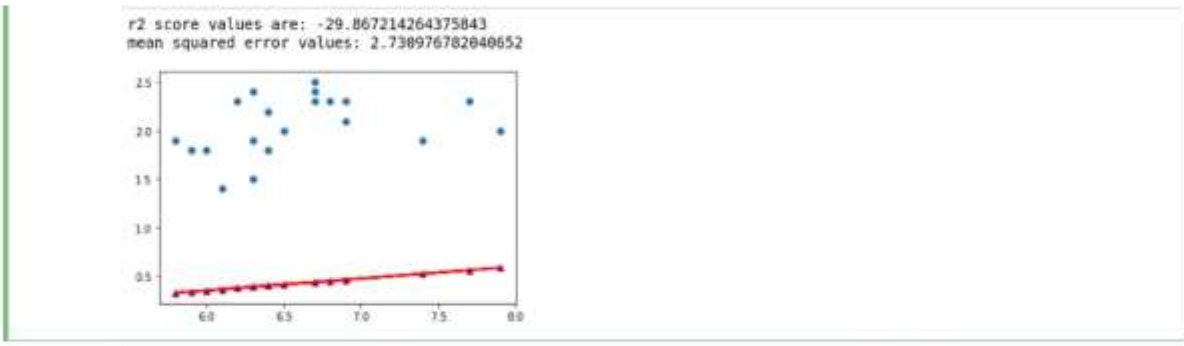
plt.scatter(X_test,Y_test)

plt.plot(X_test,Y_pred,c='red',lw=2,marker='^',markerfacecolor='blue')

print("r2 score values are:",r2_score(Y_test,Y_pred))

print("mean squared error values:",mean_squared_error(Y_test,Y_pred))

Output:

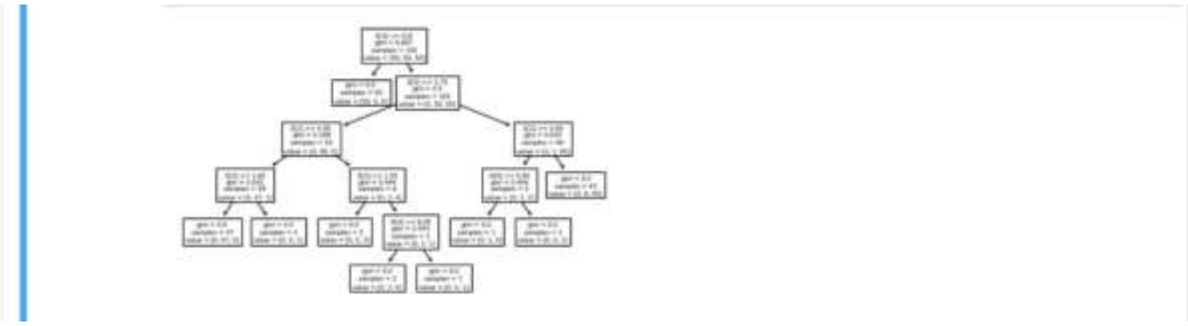


5.Build a classification model using Decision Tree algorithm on iris dataset

Programm:Using plot_tree

```
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
i=load_iris()
X,Y=i.data,i.target
m=DecisionTreeClassifier()
m=m.fit(X,Y)
m=tree.plot_tree(m)
```

Output:



Programm:export_text

```
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
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)

m=DecisionTreeClassifier()

m=m.fit(X_train,Y_train)

Y_pred=m.predict(X_test)

print("accuracy is:",accuracy_score(Y_test,Y_pred))

r=export_text(m,feature_names=i['feature_names'])

print(r)

Output:

```
accuracy is: 0.9555555555555556
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.65
|   |   |--- petal length (cm) <= 5.00
|   |   |   |--- class: 1
|   |   |--- petal length (cm) >  5.00
|   |   |   |--- sepal length (cm) <= 6.05
|   |   |   |   |--- class: 1
|   |   |   |--- sepal length (cm) >  6.05
|   |   |   |   |--- class: 2
|   |--- petal width (cm) >  1.65
|   |   |--- petal length (cm) <= 4.85
|   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |--- class: 2
|   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.85
|   |   |   |--- class: 2
```

Programm:export_graphviz

from sklearn.datasets import load_iris

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

import graphviz

i=load_iris()

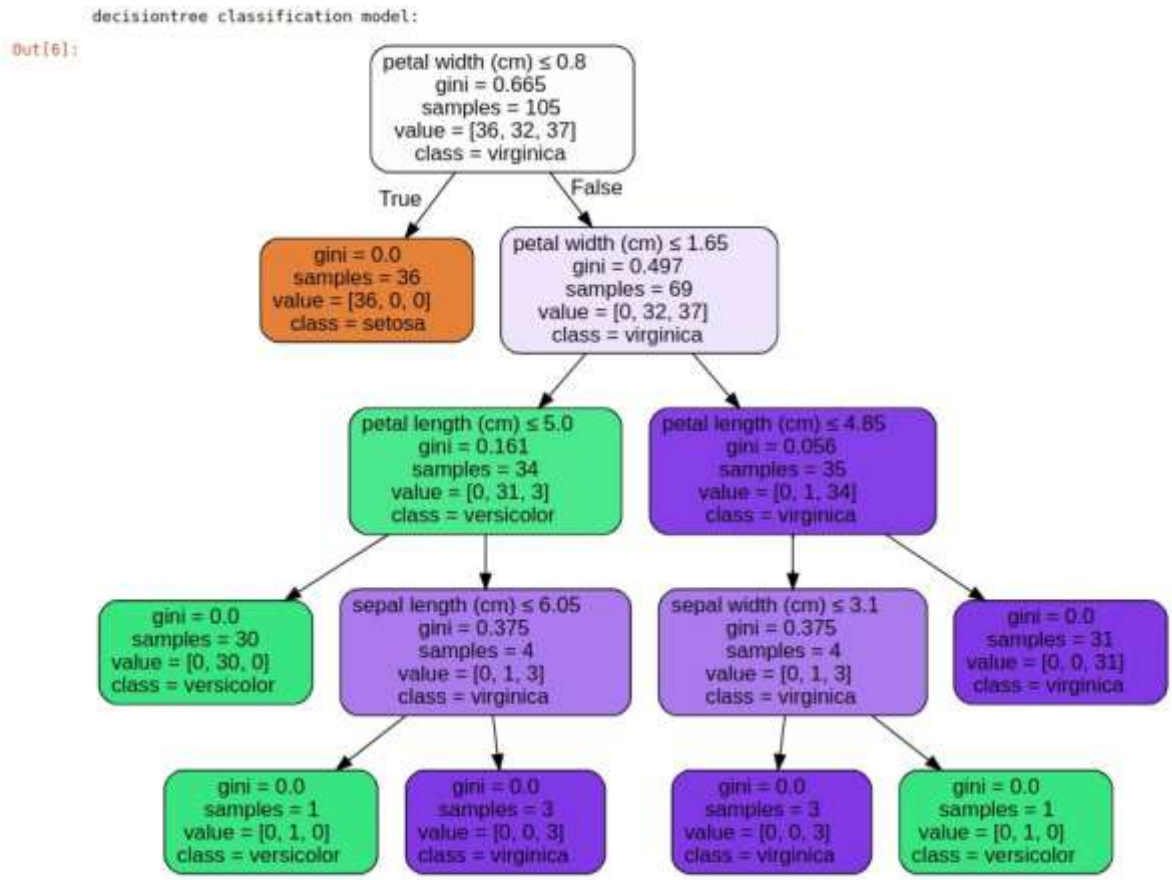
X,Y=i.data,i.target

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=1)

m=DecisionTreeClassifier()

```
m=m.fit(X_train,Y_train)

Y_pred=m.predict(X_test)

print("accuracy is:",accuracy_score(Y_test,Y_pred))

dot_data=tree.export_graphviz(m,out_file=None,filled=True,rounded=True,feature_names=i['feature_names'],class_names=['0','1','2'])

graph=graphviz.Source(dot_data)

graph
```

Output:



6. Apply Naïve Bayes Classification algorithm on any dataset

Program:

```
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
d=pd.read_csv("iris.csv")
X=d[['sepal_length','sepal_width','petal_length','petal_width']]
Y=d["species"].values
print(X)
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3)
c=GaussianNB()
c.fit(X_train,Y_train)
Y_pred=c.predict(X_test)
print(Y_pred)
accuracy=accuracy_score(Y_test,Y_pred)
print("Accuracy:",accuracy)
```

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
```

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

```
import pandas as pd
from apyori import apriori
df=pd.read_csv("Market_Basket_Optimisation .csv",header=None)
l=[]
for i in range(0,7501):
l.append([str(df.values[i,j]) for j in range(0,20)])

association_rules=apriori(l,min_support=0.0045,min_confidence=0.2,min_lift=
3,min_length=2)
association_result=list(association_rules)
for i in range(0,len(association_result)):
print(association_result[i][0])
print("------------------------------------")
for  item in association_result:
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:

```
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'})
```

```
Confidence :0.3006993006993007
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
```

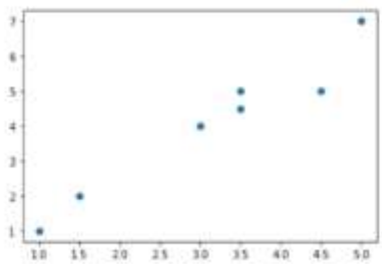
8. Apply K- Means clustering algorithm on any dataset.

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]
```



9. Apply Hierarchical Clustering algorithm on any dataset.

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=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')

print(data)

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

plt.scatter(data['a'],data['b'],c='r',marker='*')

plt.xlabel("column a")

plt.ylabel("column b")

plt.title("scatter plot of x and y")

for j in data.itertuples():

   plt.annotate(j.Index,(j.a,j.b),fontsize=15)

dist=pd.DataFrame(squareform(pdist(data[['a','b']]),'eucliedian'),colu
mns=data.index.values,index=data.index.values)

print("proximity matrix")
```

print(dist)

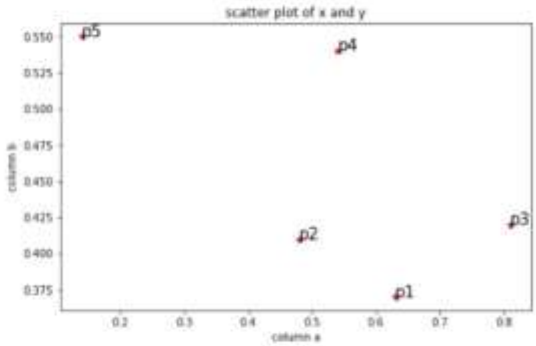
plt.figure(figsize=(12,5))

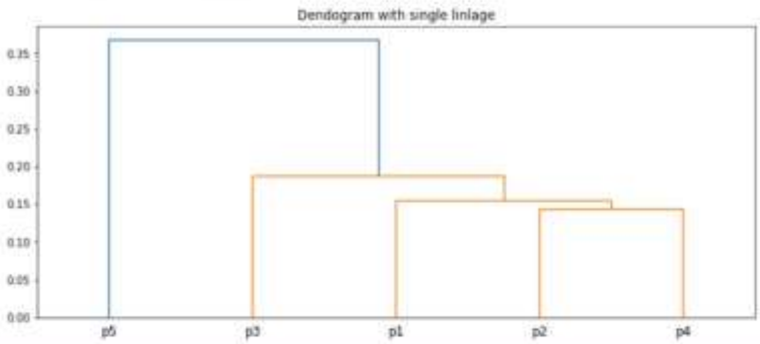
plt.title("Dendogram with single linlage")

dend=shc.dendrogram(shc.linkage(data[['a','b']],method='single'),labels=data.index)

Output:

```
point
p1      0.63  0.37
p2      0.48  0.41
p3      0.81  0.42
p4      0.54  0.54
p5      0.14  0.55
proximity matrix
          p1        p2        p3        p4        p5
p1  0.000000  0.155242  0.186815  0.192354  0.522015
p2  0.155242  0.000000  0.330151  0.143178  0.367696
p3  0.186815  0.330151  0.000000  0.295466  0.682495
p4  0.192354  0.143178  0.295466  0.000000  0.400125
p5  0.522015  0.367696  0.682495  0.400125  0.000000
```





10. Apply DBSCAN clustering algorithm on any dataset.

Program:

```
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)
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 cluster %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("cluster determined by DBSCAN")

Output:

```
[[ 0.21944999  1.43491283]
 [ 0.76205536 -1.50701033]
 [ 0.17951214  1.40902498]
 [ 0.55395946 -0.72871634]
 [ 0.84700085 -0.74057968]
 [ 0.84172206  1.80812576]
 [ 0.22960209  1.16815515]
 [ 0.76556559 -0.51520299]
 [-0.67304942 -0.80809616]
 [-0.62113428  1.97278075]
 [ 1.58928515 -0.63815665]
 [-1.61518766 -0.72879148]
 [-0.90179385 -0.6071971 ]
 [ 1.00826938 -0.97348199]
 [ 2.02971885 -0.28943621]
 [ 0.21452547  1.80296627]
 [ 1.02191358 -0.16055734]
 [-0.92808574 -0.43114313]
 [ 1.31314013 -0.81780066]
```

```
 [ 0.34993344  1.41969689]
 [ 0.71872377 -0.21881327]
 [ 0.82424386  1.23540439]
 [-1.34613815 -0.72734816]
 [-1.59870257 -0.94453715]
 [ 0.37923491 -0.59627639]
 [ 0.17013686  1.24155203]
 [-1.68900873 -0.38591244]
 [ 0.64047447 -0.36451399]
 [-1.14810577 -0.6090688 ]
 [-0.77761455 -0.79957477]
 [-0.87614308 -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 cluster 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[1]: Text(0.5, 1.0, 'cluster determined by DBSCAN')
```