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

```
#a) Loading the dataset
from sklearn.datasets import fetch_california_housing
housing_data=fetch_california_housing()
print(housing_data.data)
```

OUTPUT:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 loaddataset.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      ]]
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$
```

==

```
#b) Identifying the dependent and independent variables.
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:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 depend_and_independ.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
[5.1 3.8 1.5 0.3]  0
[5.4 3.4 1.7 0.2]  0
[5.1 3.7 1.5 0.4]  0
```

```python
#c) Dealing with missing data
import numpy as np
import pandas as pd
df=pd.read_csv('stu.csv')
print("Before filling missing data")
print(df)
df['MARKS2']=df['MARKS2'].fillna(df['MARKS2'].mean())
print("After filling missing data")
print(df)
```

OUTPUT:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 dealing_with_missing_data.py
Before filling missing data
    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         Kusuma'      90    60.0      70   'Excellent'  'Femlae'
8      110          Kamal'      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'
After filling missing data
    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         Kusuma'      90  60.000000      70   'Excellent'  'Femlae'
8      110          Kamal'      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'
```

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

a.Dealing with Categorical Data

```
#a)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:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 categorical_data.py
0           'Akash'
1            'Ajay'
2         'Brijesh'
3         'Chandra'
4            'Ramu'
5            'Devi'
6         'Lakshmi'
7           Kusuma'
8            Kamal'
9       'SimplyRam'
10      SimplyRamu'
11      SimplyRams'
12     SimplyRamos'
Name: Name, dtype: object
0      1
1      0
2      2
3      3
4      6
5      4
6      5
7      9
8      8
9      7
10    12
11    11
12    10
Name: Name, dtype: int64
```

==

```
#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:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 label_binarizer.py
0           'Akash'
1            'Ajay'
2         'Brijesh'
3         'Chandra'
4            'Ramu'
5            'Devi'
6         'Lakshmi'
7           Kusuma'
8            Kamal'
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 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]
 [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 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 0 0 0 0 1 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 1 0 0 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 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0]]
```

==

```
#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'])
```

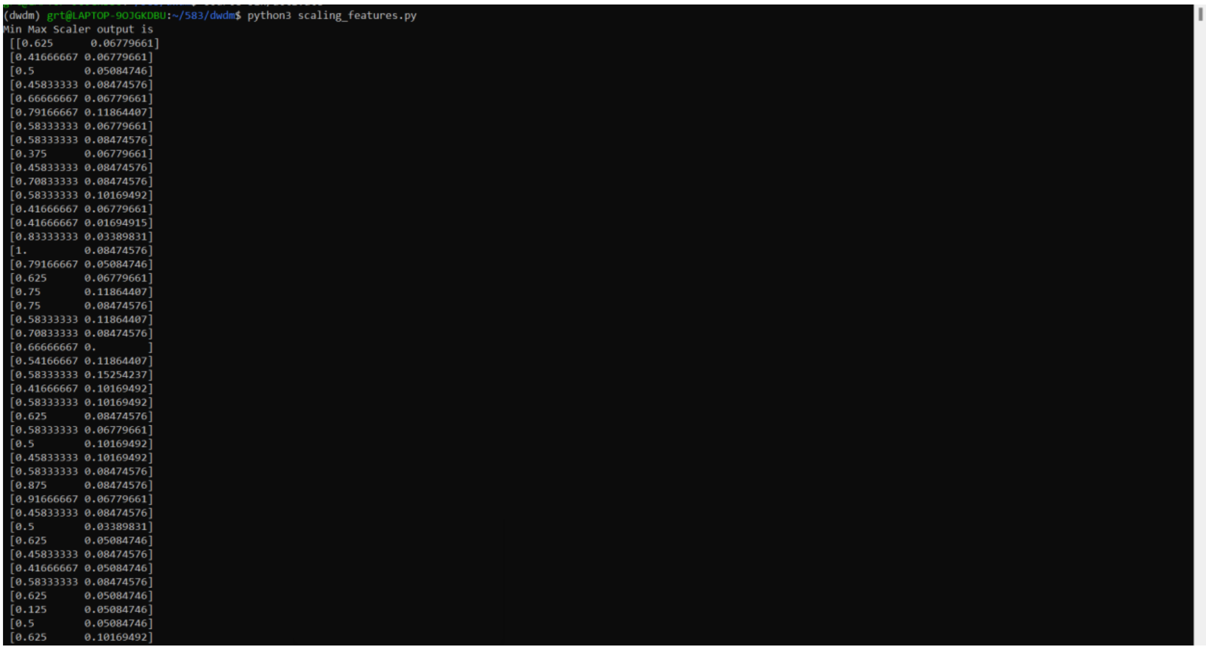
OUTPUT:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 astype.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      6
5      4
6      5
7      9
8      8
9      7
10    12
11    11
12    10
Name: Name, dtype: int8
```

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:



```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 scaling_features.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 0.06779661]
 [0.5        0.10169492]
 [0.45833333 0.10169492]
 [0.58333333 0.08474576]
 [0.875      0.08474576]
 [0.91666667 0.06779661]
 [0.45833333 0.08474576]
 [0.5        0.03389831]
 [0.625      0.05084746]
 [0.45833333 0.08474576]
 [0.41666667 0.05084746]
 [0.58333333 0.08474576]
 [0.625      0.05084746]
 [0.125      0.05084746]
 [0.5        0.05084746]
 [0.625      0.10169492]
```

c.Splitting dataset into Testing and Trainig sets

```
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:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 train_and_test.py
X training set values   Y training set values
[7.7 2.6 6.9 2.3]        2
[5.7 3.8 1.7 0.3]        0
[5.  3.6 1.4 0.2]        0
[4.8 3.  1.4 0.3]        0
[5.2 2.7 3.9 1.4]        1
[5.1 3.4 1.5 0.2]        0
[5.5 3.5 1.3 0.2]        0
[7.7 3.8 6.7 2.2]        2
[6.9 3.1 5.4 2.1]        2
[7.3 2.9 6.3 1.8]        2
[6.4 2.8 5.6 2.2]        2
[6.2 2.8 4.8 1.8]        2
[6.  3.4 4.5 1.6]        1
[7.7 2.8 6.7 2. ]        2
[5.7 3.  4.2 1.2]        1
[4.8 3.4 1.6 0.2]        0
[5.7 2.5 5.  2. ]        2
[6.3 2.7 4.9 1.8]        2
[4.8 3.  1.4 0.1]        0
[4.7 3.2 1.3 0.2]        0
[6.5 3.  5.8 2.2]        2
[4.6 3.4 1.4 0.3]        0
[6.1 3.  4.9 1.8]        2
[6.5 3.2 5.1 2. ]        2
[6.7 3.1 4.4 1.4]        1
[5.7 2.8 4.5 1.3]        1
[6.7 3.3 5.7 2.5]        2
[6.  3.  4.8 1.8]        2
[5.1 3.8 1.6 0.2]        0
[6.  2.2 4.  1. ]        1
[6.4 2.9 4.3 1.3]        1
[6.5 3.  5.5 1.8]        2
[5.  2.3 3.3 1. ]        1
[6.3 3.3 6.  2.5]        2
[5.5 2.5 4.  1.3]        1
[5.4 3.7 1.5 0.2]        0
[4.9 3.1 1.5 0.2]        0
[5.2 4.1 1.5 0.1]        0
[6.7 3.3 5.7 2.1]        2
[4.4 3.  1.3 0.2]        0
[6.  2.7 5.1 1.6]        1
[6.4 2.7 5.3 1.9]        2
[5.9 3.  5.1 1.8]        2
[5.2 3.5 1.5 0.2]        0
```

==

3.Splitting the following Simiarity and Dissimilarity measures using python

```python
#a) 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------")
print("pearson correlation:",corr)
print(end="\n")

#b)Cosine Similarity
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()
print("----------Cosine Similarity---------")
df=pd.DataFrame(Y,columns=ob.get_feature_names_out(),index=['A','B'])
print(df)
print("similarity matrix:\n",cosine_similarity(df,df))
print(end="\n")
```

```
#c) Jaccard Similarity
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---------")
print(" jaccard distance:",jaccard(a,b))
print(end="\n")

#d) Manhattan Distance
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)
print("-------Manhattan Distance-------")
print(manhattan_distances(X,Y,sum_over_features=False))
print(end="\n")

#e) Euclidean Distance
from sklearn.metrics.pairwise import euclidean_distances
X=[[0,1],[1,1]]
print("------Euclidean Distance-------")
print(euclidean_distances(X,X))
print("get distance from origin")
print(euclidean_distances(X,[[0,0]]))
```

OUTPUT:

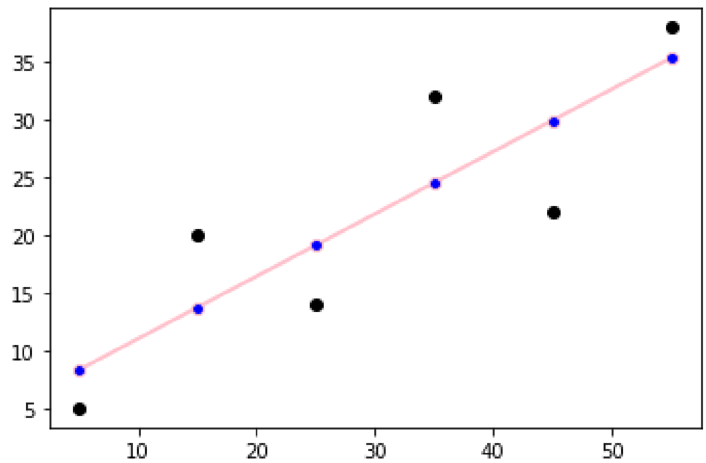
```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 similarity_and_dissimilarity.py
-------Pearson Correlation------
pearson correlation: (-0.7000000000000001, 0.1881204043741873)

----------Cosine Similarity---------
   be  can  deep  hard  learning  soft
A   1    1     1     1         1     0
B   1    1     1     0         1     1
similarity matrix:
 [[1.  0.8]
 [0.8 1. ]]

---------Jaccard Distance---------
 jaccard distance: 0.8

-------Manhattan Distance-------
[[1. 1.]
 [1. 1.]]

------Euclidean Distance-------
[[0. 1.]
 [1. 0.]]
get distance from origin
[[1.        ]
 [1.41421356]]
```

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

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
x=np.array([5,15,25,35,45,55]).reshape((-1,1))
y=np.array([5,20,14,32,22,38])
print(x,y)
model=LinearRegression()
model.fit(x,y)
result=model.score(x,y)
print("Score:",result)
print("Intercept:",model.intercept_)
print("Slope:",model.coef_)
y_pred=model.predict(x)
print("Actual Values of y:",y)
print("Predicted Values of y:",y_pred)
plt.scatter(x,y,color="Black")
plt.plot(x,y_pred,color="Pink",linewidth=2,marker='o',markerfacecolor="Blue")
```

OUTPUT:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 linearregression.py
[[ 5]
 [15]
 [25]
 [35]
 [45]
 [55]] [ 5 20 14 32 22 38]
Score: 0.7158756137479542
Intercept: 5.633333333333329
Slope: [0.54]
Actual Values of y: [ 5 20 14 32 22 38]
Predicted Values of y: [ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```



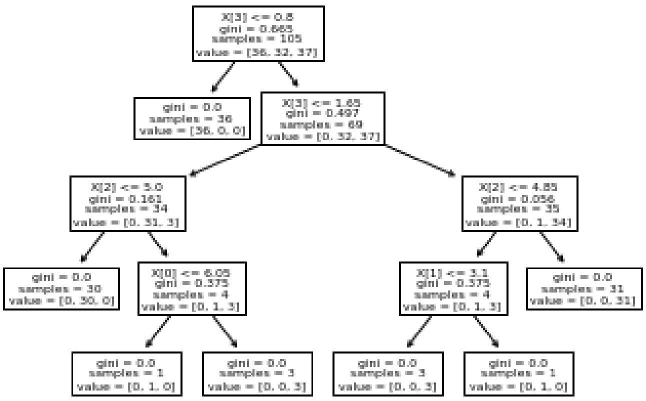
5.Build a classification model using Decision Tree algorithm on iris dataset

```
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 import metrics
import graphviz

#step1 load iris data set
ir = load_iris()

#step 2:Divide dependent and independent variables
X, y = ir.data, ir.target

#step 3: split train data and split data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=
1)

#step4: crate an object for decision tree
clf=tree.DecisionTreeClassifier()
print(clf)

#step5: fit the data
clf=clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test,y_pred))

#By using export_tree
print("-------By using export_tree-------")
r=export_text(clf,feature_names=ir['feature_names'])
print(r)

#By using plot_tree
print("-------By using plot_tree--------")
clf=tree.plot_tree(clf)

#By using export_graphviz
print("---------By using export_graphviz--------")
dot_data=tree.export_graphviz(clf,out_file=None,filled=True,rounded=True,featu
re_names=ir['feature_names'],class_names=['0','1','2'])
graph=graphviz.Source(dot_data)
graph
```

OUTPUT:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 DecisionTree.py
DecisionTreeClassifier()
Accuracy: 0.9555555555555556
-------By using export_tree-------
|--- 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
```

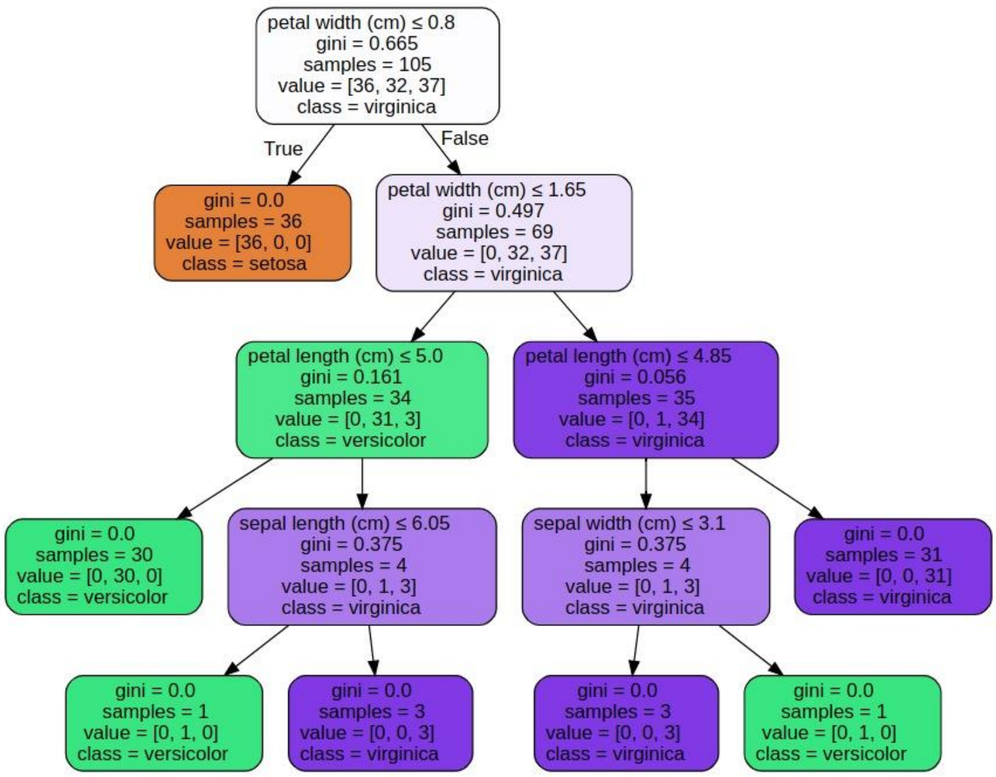
-------By using plot_tree--------



---------By using export_graphviz--------

decisiontree classification model:

Out[6]:



6.Apply Naïve Bayes Classification algorithm on any dataset

```
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:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 naive_bayes.py
     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]
['setosa' 'virginica' 'virginica' 'setosa' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'virginica' 'setosa' 'versicolor' 'versicolor' 'virginica' 'virginica'
 'virginica' 'setosa' 'setosa' 'versicolor' 'virginica' 'virginica'
 'versicolor' 'virginica' 'virginica' 'virginica' 'setosa' 'virginica'
 'versicolor' 'virginica' 'setosa' 'setosa' 'versicolor' 'virginica'
 'virginica' 'versicolor' 'setosa' 'setosa' 'virginica' 'virginica'
 'setosa' 'setosa' 'versicolor']
Accuracy: 0.9333333333333333
```

===

8.Apply K-Means clustering algorithm on any dataset

```
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)
plt.scatter(X[:,0],X[:,1])
Kmeans=KMeans(n_clusters=2)
Kmeans.fit(X)
```

OUTPUT:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 kmeans_algorithm.py
[[1.  1. ]
 [1.5 2. ]
 [3.  4. ]
 [5.  7. ]
 [3.5 5. ]
 [4.5 5. ]
 [3.5 4.5]]
```

==

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

```python
import pandas as pd
from apyori import apriori
st_df=pd.read_csv("Market_Basket_Optimisation.csv",header=None)
st_df
#converting dataframe into list of lists
l=[]
for i in range(1,7501):
    l.append([str(st_df.values[i,j]) for j in range (0,20)])

#applying apriori algorithm
association_rules=apriori(l,min_support=0.0045,min_confidance=0.2,min_lift=3,min_length=2)
association_results=list(association_rules)

for i in range(0,len(association_results)):
    print(association_results[i][0])

for item in association_results:
    #first index of the inner list
    #Contains base item and add item
    pair=item[0]
    items=[x for x in pair]
    print("Rule:" + items[0] + "-> " + items[1])
    #second index of the inner list
    print("Support:" +str(item[1]))
    #third index of the list located at 0th poisition
    #of the third index of the inner list
    print("Confidence: "+str(item[2][0][2]))
    print("Lift:" + str(item[2][0][3]))
    print("-----------------------------------------")
```

OUTPUT:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 apriori_algorithm.py
frozenset({'chicken', 'light cream'})
frozenset({'mushroom cream sauce', 'escalope'})
frozenset({'pasta', 'escalope'})
frozenset({'herb & pepper', 'ground beef'})
frozenset({'tomato sauce', 'ground beef'})
frozenset({'whole wheat pasta', 'olive oil'})
frozenset({'shrimp', 'pasta'})
frozenset({'nan', 'chicken', 'light cream'})
frozenset({'frozen vegetables', 'shrimp', 'chocolate'})
frozenset({'spaghetti', 'ground beef', 'cooking oil'})
frozenset({'mushroom cream sauce', 'nan', 'escalope'})
frozenset({'pasta', 'nan', 'escalope'})
frozenset({'spaghetti', 'ground beef', 'frozen vegetables'})
frozenset({'milk', 'olive oil', 'frozen vegetables'})
frozenset({'shrimp', 'mineral water', 'frozen vegetables'})
frozenset({'spaghetti', 'olive oil', 'frozen vegetables'})
frozenset({'spaghetti', 'shrimp', 'frozen vegetables'})
frozenset({'spaghetti', 'tomatoes', 'frozen vegetables'})
frozenset({'spaghetti', 'ground beef', 'grated cheese'})
frozenset({'herb & pepper', 'ground beef', 'mineral water'})
frozenset({'herb & pepper', 'ground beef', 'nan'})
frozenset({'herb & pepper', 'ground beef', 'spaghetti'})
frozenset({'ground beef', 'milk', 'olive oil'})
frozenset({'tomato sauce', 'ground beef', 'nan'})
frozenset({'spaghetti', 'ground beef', 'shrimp'})
frozenset({'milk', 'soup', 'mineral water'})
frozenset({'spaghetti', 'milk', 'olive oil'})
frozenset({'olive oil', 'soup', 'mineral water'})
frozenset({'whole wheat pasta', 'nan', 'olive oil'})
frozenset({'nan', 'shrimp', 'pasta'})
frozenset({'spaghetti', 'pancakes', 'olive oil'})
frozenset({'frozen vegetables', 'shrimp', 'nan', 'chocolate'})
frozenset({'spaghetti', 'ground beef', 'cooking oil', 'nan'})
frozenset({'spaghetti', 'ground beef', 'nan', 'frozen vegetables'})
frozenset({'spaghetti', 'milk', 'mineral water', 'frozen vegetables'})
frozenset({'milk', 'nan', 'olive oil', 'frozen vegetables'})
frozenset({'nan', 'shrimp', 'mineral water', 'frozen vegetables'})
frozenset({'spaghetti', 'nan', 'olive oil', 'frozen vegetables'})
frozenset({'spaghetti', 'nan', 'shrimp', 'frozen vegetables'})
frozenset({'spaghetti', 'tomatoes', 'nan', 'frozen vegetables'})
frozenset({'spaghetti', 'ground beef', 'nan', 'grated cheese'})
frozenset({'herb & pepper', 'ground beef', 'nan', 'mineral water'})
frozenset({'herb & pepper', 'ground beef', 'nan', 'spaghetti'})
frozenset({'ground beef', 'milk', 'nan', 'olive oil'})
frozenset({'spaghetti', 'ground beef', 'nan', 'shrimp'})
```

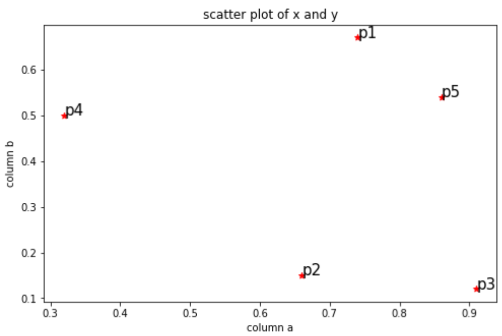
```
frozenset({'olive oil', 'nan', 'soup', 'mineral water'})
frozenset({'spaghetti', 'nan', 'pancakes', 'olive oil'})
frozenset({'milk', 'spaghetti', 'nan', 'frozen vegetables', 'mineral water'})
Rule:chicken-> light cream
Support:0.004533333333333334
Confidence: 0.07555555555555557
Lift:4.8433048433048445
-----------------------------------------
Rule:mushroom cream sauce-> escalope
Support:0.005733333333333333
Confidence: 0.0722689075630252
Lift:3.790327319739084
-----------------------------------------
Rule:pasta-> escalope
Support:0.005866666666666667
Confidence: 0.07394957983193277
Lift:4.700185158809287
-----------------------------------------
Rule:herb & pepper-> ground beef
Support:0.016
Confidence: 0.1628222523744912
Lift:3.2915549671393096
-----------------------------------------
Rule:tomato sauce-> ground beef
Support:0.005333333333333333
Confidence: 0.05427408412483039
Lift:3.8401474616625277
-----------------------------------------
Rule:whole wheat pasta-> olive oil
Support:0.008
Confidence: 0.12170385395537525
Lift:4.130221288078346
-----------------------------------------
Rule:shrimp-> pasta
Support:0.005066666666666666
Confidence: 0.3220338983050848
Lift:4.514493901473151
-----------------------------------------
Rule:nan-> chicken
Support:0.004533333333333334
Confidence: 0.07555555555555557
Lift:4.8433048433048445
-----------------------------------------
Rule:frozen vegetables-> shrimp
Support:0.005333333333333333
Confidence: 0.05594405594405594
Lift:3.108003108003108
```

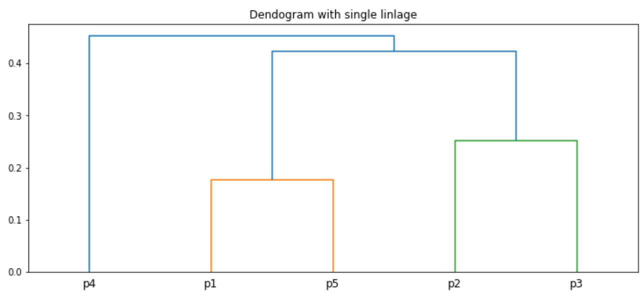
9.Apply Hierarchial Clustering algorithm on any dataset

```
import numpy as np
import pandas as pd
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=(5,3))
plt.scatter(data['a'],data['b'],c='r',marker='o')
plt.xlabel('column a')
plt.ylabel('column b')
plt.title('scatter plot x and y')
for j in data.itertuples():
    plt.annotate(j.Index,(j.a,j.b),fontsize=15)
```

OUTPUT:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 hierarchial_clustering.py
          a     b
point
p1     0.35  0.54
p2     0.22  0.22
p3     0.41  0.60
p4     0.37  0.82
p5     0.72  0.18
proximity matrix
          p1        p2        p3        p4        p5
p1  0.000000  0.345398  0.084853  0.280713  0.516236
p2  0.345398  0.000000  0.424853  0.618466  0.501597
p3  0.084853  0.424853  0.000000  0.223607  0.522015
p4  0.280713  0.618466  0.223607  0.000000  0.729452
p5  0.516236  0.501597  0.522015  0.729452  0.000000
```





10.Apply dbscan algorithm on any dataset

```
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:

```
(dwdm) grt@LAPTOP-9OJGKDBU:~/583/dwdm$ python3 dbscan.py
[[ 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]
 [-1.51911189 -0.79773621]
 [-1.55314151 -0.56212295]
 [-0.08122475  0.90177102]
 [-1.11135974 -0.41252607]
 [-1.02371315 -0.98077793]
 [-1.04447701 -0.50489449]
 [ 0.53987738  1.05173636]
 [-1.38751061 -0.32845505]
 [ 0.99291866  0.97132263]
 [-1.73164148 -0.43845562]
 [-0.35493331 -1.20752094]
 [ 0.70510875  1.71681314]
 [ 1.17456328  0.8081482 ]
 [-1.73524259 -0.08179708]
 [ 1.33510777 -1.29449661]
 [ 1.40129845 -0.46490155]
 [ 0.60100499 -1.27490336]
 [ 0.46883888 -0.65522071]
 [-0.2177715   0.71514667]
 [ 0.58563711 -0.51343774]
 [ 2.12903658 -0.42352534]
 [-1.00353943  1.52912482]
 [ 1.56155776 -1.11591808]
 [ 0.45842855 -0.90442994]
 [ 1.46862236 -0.63754612]
```

```
 [-0.15423336  1.18259797]
 [ 1.28906178 -0.80523052]
 [ 0.102854    1.44600664]
 [-0.57941603  1.23281117]
 [-0.41633082  1.57149166]
 [-1.5943798  -0.09327404]
 [-0.25148103  0.62809738]
 [-1.03541767 -1.122577  ]
 [-1.50997304 -0.73567119]
 [ 1.12800861 -0.52936333]
 [ 1.48152185 -0.7166135 ]
 [ 0.49317939  1.3471885 ]
 [ 0.89285887 -1.21682128]
 [ 0.53450078 -0.89630194]
 [-0.08070404  1.15574202]
 [ 0.82780654 -0.08661587]
 [ 0.34993344  1.41969689]
 [ 0.71872377 -0.21881327]
 [ 0.82424386  1.23540439]
 [-1.34613015 -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]
```