Hierarchical K-Means

Name - Amith Korada

Course - AI & ML
(Batch - 4)

Duration - 12 Months

Problem Statement - Perform Hierarchical Clustering from scratch and also using sklearn to perform wholesale customer segmentation based on their annual spending on products

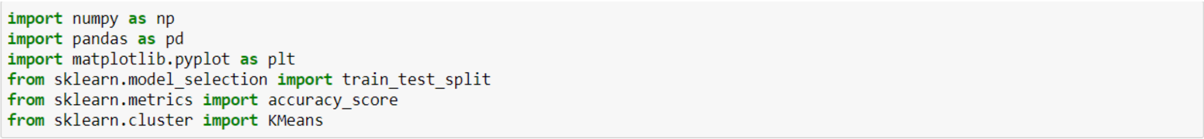
Prerequisites -

What things you need to install the software and how to install them:

Python 3.6 This setup requires that your machine has the latest version of python. The following URL https://www.python.org/downloads/ can be referred to as download python.

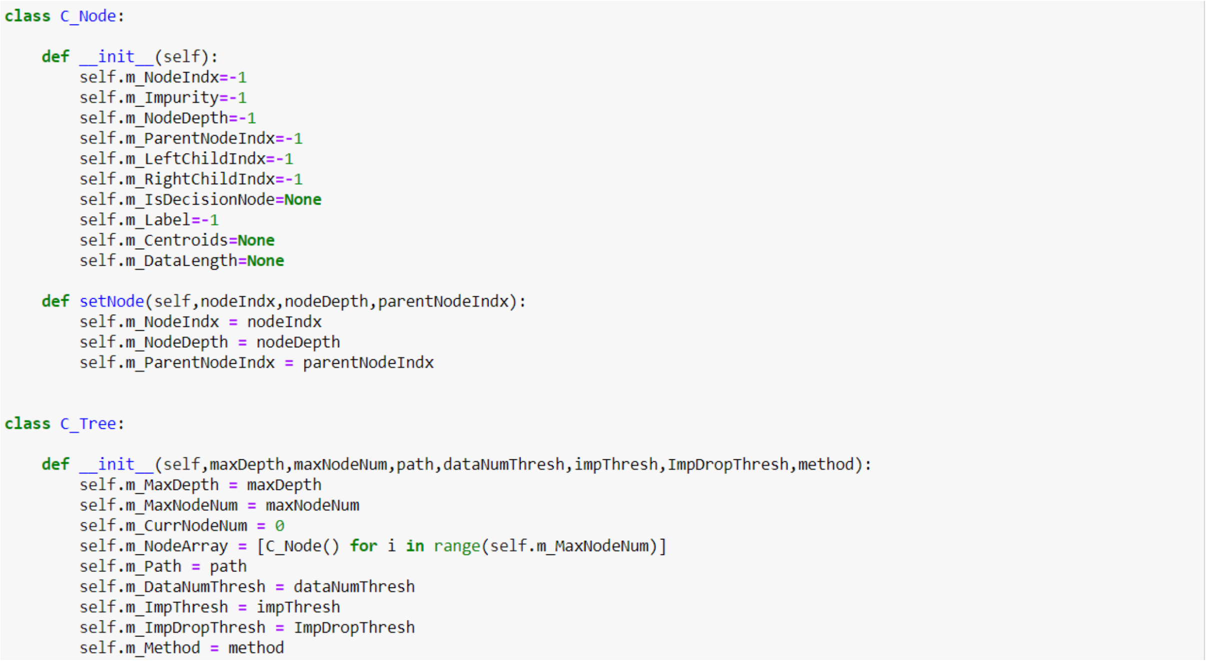
The second and easier option is to download anaconda and use its anaconda prompt to run the commands. To install anaconda check this URL https://www.anaconda.com/download/
You will also need to download and install the below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.6 then run the below commands in command prompt/terminal to install these packages pip install -U sci-kit-learn pip install NumPy pip install scipy if you have chosen to install anaconda then run the below commands in anaconda prompt to install these packages conda install -c sci-kit-learn conda install -c anaconda numpy conda install -c anaconda scipy.

1. Importing necessary libraries-



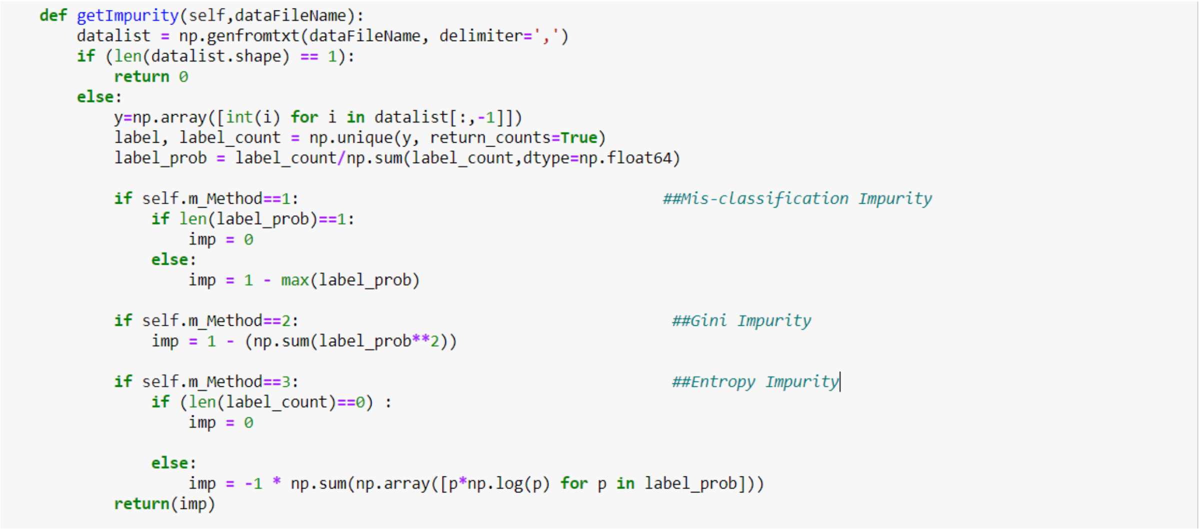
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans
```

2. Defining Node and Tree classes -



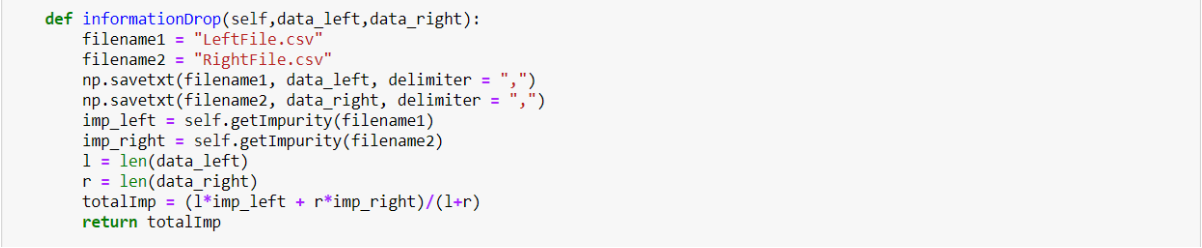
```
class C_Node:

    def __init__(self):
        self.m_NodeIndx=-1
        self.m_Impurity=-1
        self.m_NodeDepth=-1
        self.m_ParentNodeIndx=-1
        self.m_LeftChildIndx=-1
        self.m_RightChildIndx=-1
        self.m_IsDecisionNode=None
        self.m_Label=-1
        self.m_Centroids=None
        self.m_DataLength=None

    def setNode(self,nodeIndx,nodeDepth,parentNodeIndx):
        self.m_NodeIndx = nodeIndx
        self.m_NodeDepth = nodeDepth
        self.m_ParentNodeIndx = parentNodeIndx


class C_Tree:

    def __init__(self,maxDepth,maxNodeNum,path,dataNumThresh,impThresh,ImpDropThresh,method):
        self.m_MaxDepth = maxDepth
        self.m_MaxNodeNum = maxNodeNum
        self.m_CurrNodeNum = 0
        self.m_NodeArray = [C_Node() for i in range(self.m_MaxNodeNum)]
        self.m_Path = path
        self.m_DataNumThresh = dataNumThresh
        self.m_ImpThresh = impThresh
        self.m_ImpDropThresh = ImpDropThresh
        self.m_Method = method
```

3. Function under Tree class to calculate the impurity-



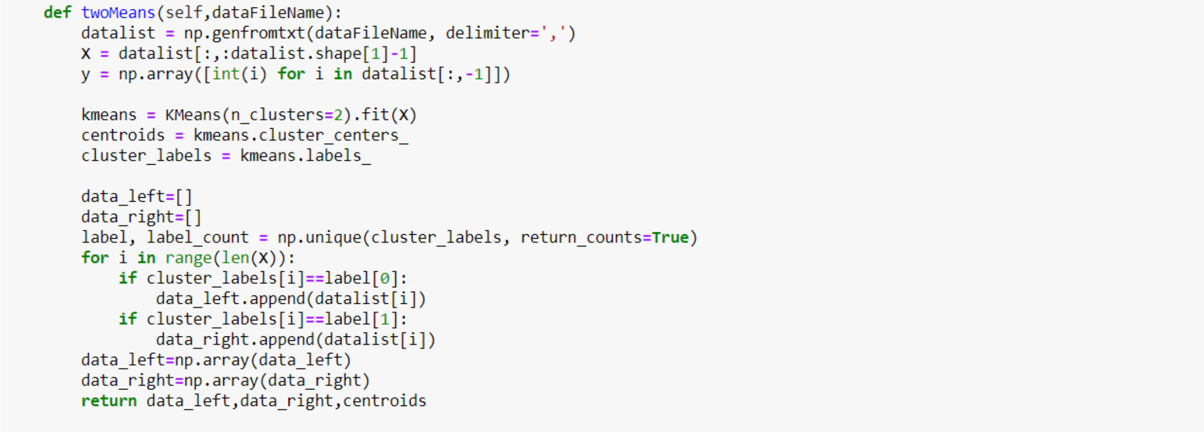
```
    def getImpurity(self,dataFileName):
        datalist = np.genfromtxt(dataFileName, delimiter=',')
        if (len(datalist.shape) == 1):
            return 0
        else:
            y=np.array([int(i) for i in datalist[:,-1]])
            label, label_count = np.unique(y, return_counts=True)
            label_prob = label_count/np.sum(label_count,dtype=np.float64)

            if self.m_Method==1:                                    ##Mis-classification Impurity
                if len(label_prob)==1:
                    imp = 0
                else:
                    imp = 1 - max(label_prob)

            if self.m_Method==2:                                     ##Gini Impurity
                imp = 1 - (np.sum(label_prob**2))

            if self.m_Method==3:                                     ##Entropy Impurity
                if (len(label_count)==0) :
                    imp = 0

                else:
                    imp = -1 * np.sum(np.array([p*np.log(p) for p in label_prob]))
            return(imp)
```

4. Function of Tree class to calculate information drop-



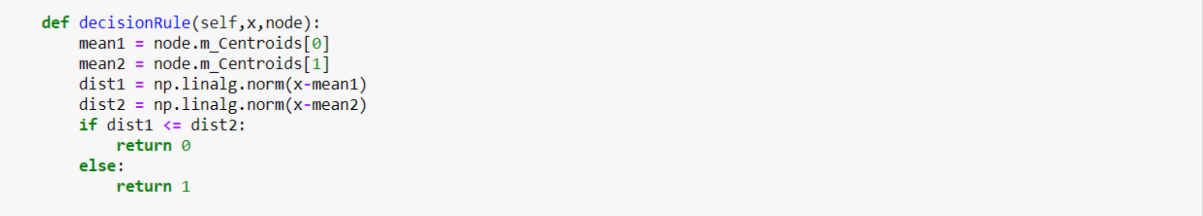
```
    def informationDrop(self,data_left,data_right):
        filename1 = "LeftFile.csv"
        filename2 = "RightFile.csv"
        np.savetxt(filename1, data_left, delimiter = ",")
        np.savetxt(filename2, data_right, delimiter = ",")
        imp_left = self.getImpurity(filename1)
        imp_right = self.getImpurity(filename2)
        l = len(data_left)
        r = len(data_right)
        totalImp = (l*imp_left + r*imp_right)/(l+r)
        return totalImp
```

5. 2-Means Classification-



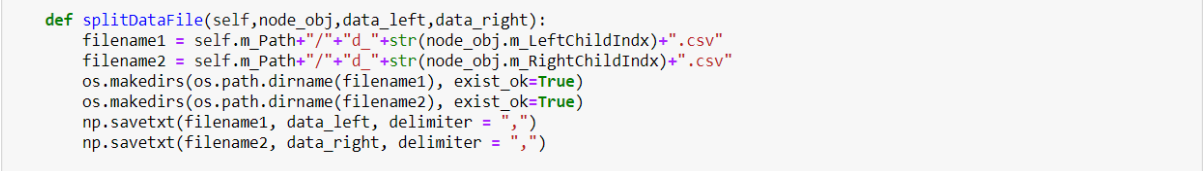
```python
def twoMeans(self,dataFileName):
    datalist = np.genfromtxt(dataFileName, delimiter=',')
    X = datalist[:,:datalist.shape[1]-1]
    y = np.array([int(i) for i in datalist[:,-1]])

    kmeans = KMeans(n_clusters=2).fit(X)
    centroids = kmeans.cluster_centers_
    cluster_labels = kmeans.labels_

    data_left=[]
    data_right=[]
    label, label_count = np.unique(cluster_labels, return_counts=True)
    for i in range(len(X)):
        if cluster_labels[i]==label[0]:
            data_left.append(datalist[i])
        if cluster_labels[i]==label[1]:
            data_right.append(datalist[i])
    data_left=np.array(data_left)
    data_right=np.array(data_right)
    return data_left,data_right,centroids
```

6. Decision rule to assign data point to right/left child -



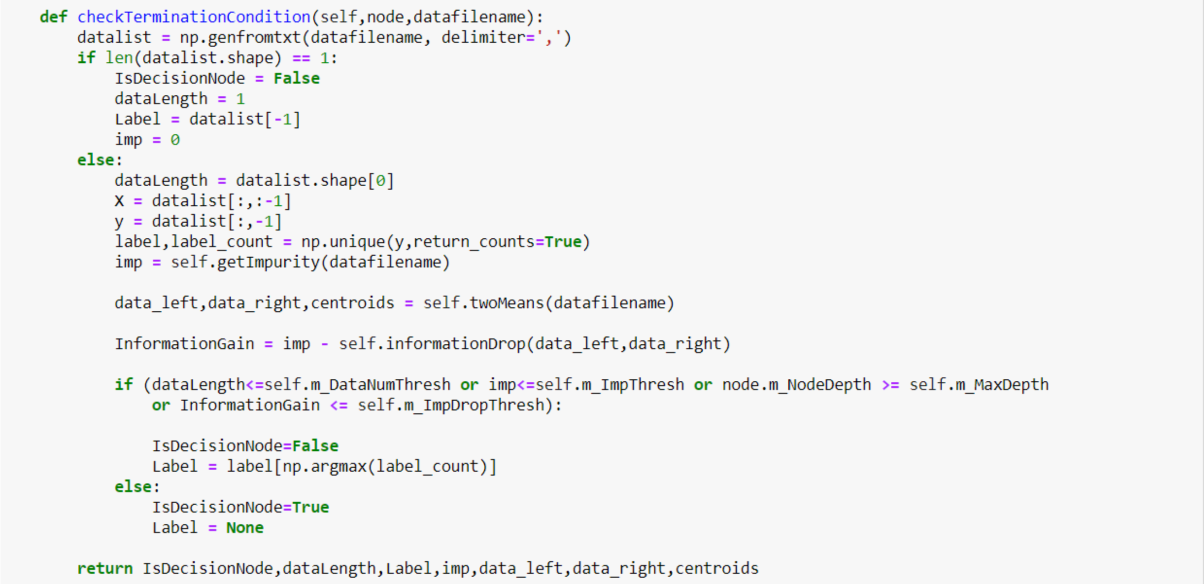
```python
def decisionRule(self,x,node):
    mean1 = node.m_Centroids[0]
    mean2 = node.m_Centroids[1]
    dist1 = np.linalg.norm(x-mean1)
    dist2 = np.linalg.norm(x-mean2)
    if dist1 <= dist2:
        return 0
    else:
        return 1
```

7. Splitting the data-



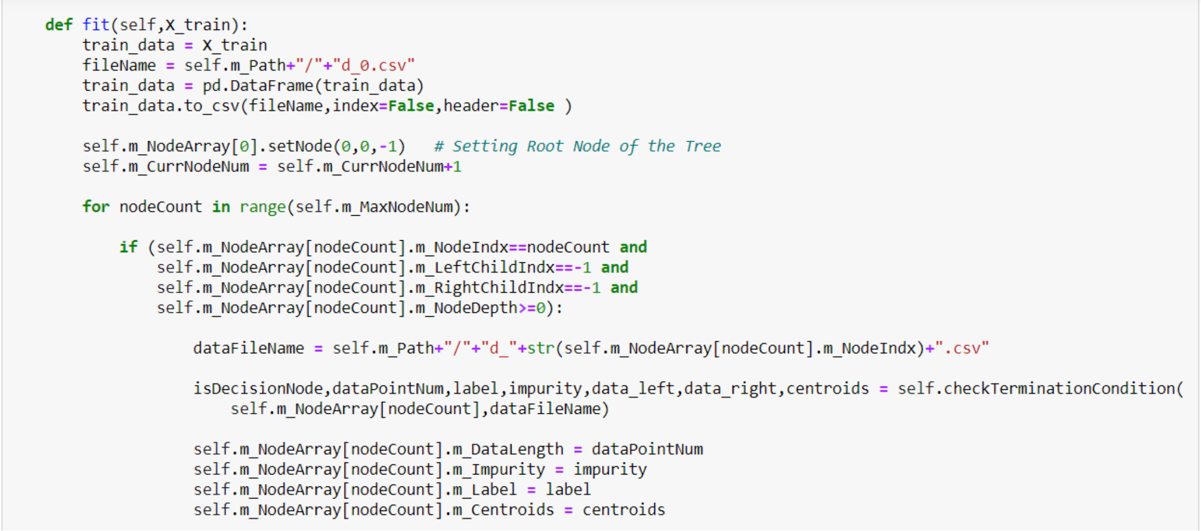
```python
def splitDataFile(self,node_obj,data_left,data_right):
    filename1 = self.m_Path+"/"+"d_"+str(node_obj.m_LeftChildIndx)+".csv"
    filename2 = self.m_Path+"/"+"d_"+str(node_obj.m_RightChildIndx)+".csv"
    os.makedirs(os.path.dirname(filename1), exist_ok=True)
    os.makedirs(os.path.dirname(filename2), exist_ok=True)
    np.savetxt(filename1, data_left, delimiter = ",")
    np.savetxt(filename2, data_right, delimiter = ",")
```

8. Checking Termination condition-

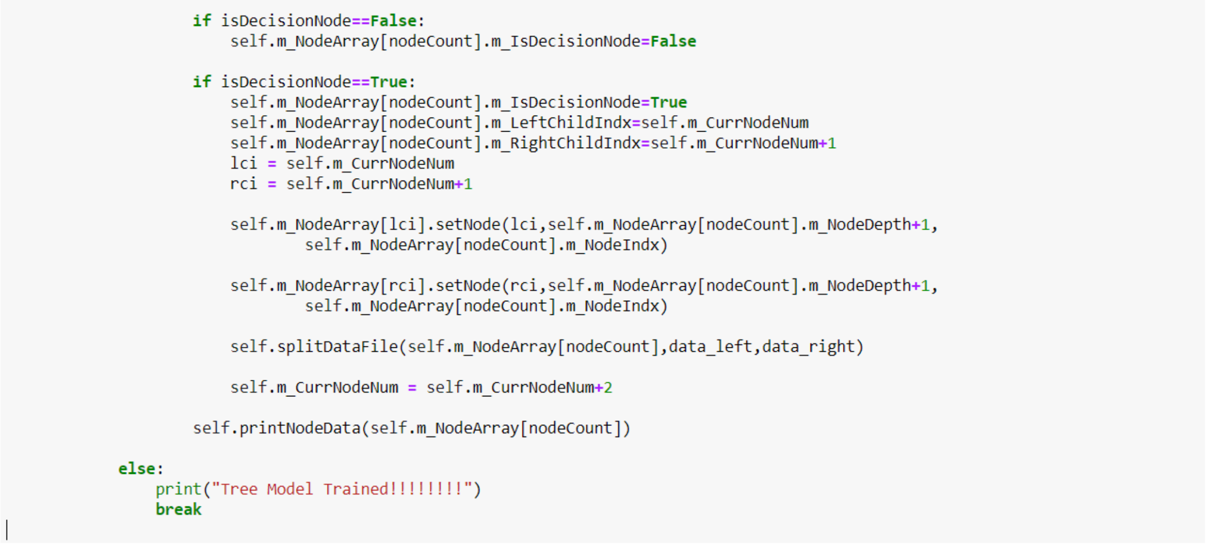


```python
def checkTerminationCondition(self,node,datafilename):
    datalist = np.genfromtxt(datafilename, delimiter=',')
    if len(datalist.shape) == 1:
        IsDecisionNode = False
        dataLength = 1
        Label = datalist[-1]
        imp = 0
    else:
        dataLength = datalist.shape[0]
        X = datalist[:,:-1]
        y = datalist[:,-1]
        label,label_count = np.unique(y,return_counts=True)
        imp = self.getImpurity(datafilename)

        data_left,data_right,centroids = self.twoMeans(datafilename)

        InformationGain = imp - self.informationDrop(data_left,data_right)

        if (dataLength<=self.m_DataNumThresh or imp<=self.m_ImpThresh or node.m_NodeDepth >= self.m_MaxDepth
            or InformationGain <= self.m_ImpDropThresh):

            IsDecisionNode=False
            Label = label[np.argmax(label_count)]
        else:
            IsDecisionNode=True
            Label = None

    return IsDecisionNode,dataLength,Label,imp,data_left,data_right,centroids
```

9. Fitting the model-

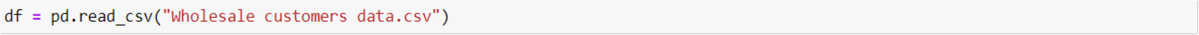


```
    def fit(self,X_train):
        train_data = X_train
        fileName = self.m_Path+"/"+"d_0.csv"
        train_data = pd.DataFrame(train_data)
        train_data.to_csv(fileName,index=False,header=False )

        self.m_NodeArray[0].setNode(0,0,-1)   # Setting Root Node of the Tree
        self.m_CurrNodeNum = self.m_CurrNodeNum+1

        for nodeCount in range(self.m_MaxNodeNum):

            if (self.m_NodeArray[nodeCount].m_NodeIndx==nodeCount and
                self.m_NodeArray[nodeCount].m_LeftChildIndx==-1 and
                self.m_NodeArray[nodeCount].m_RightChildIndx==-1 and
                self.m_NodeArray[nodeCount].m_NodeDepth>=0):

                dataFileName = self.m_Path+"/"+"d_"+str(self.m_NodeArray[nodeCount].m_NodeIndx)+".csv"

                isDecisionNode,dataPointNum,label,impurity,data_left,data_right,centroids = self.checkTerminationCondition(
                    self.m_NodeArray[nodeCount],dataFileName)

                self.m_NodeArray[nodeCount].m_DataLength = dataPointNum
                self.m_NodeArray[nodeCount].m_Impurity = impurity
                self.m_NodeArray[nodeCount].m_Label = label
                self.m_NodeArray[nodeCount].m_Centroids = centroids
```

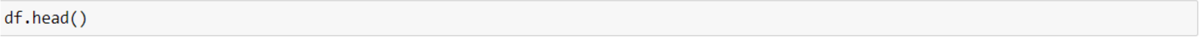


```
                if isDecisionNode==False:
                    self.m_NodeArray[nodeCount].m_IsDecisionNode=False

                if isDecisionNode==True:
                    self.m_NodeArray[nodeCount].m_IsDecisionNode=True
                    self.m_NodeArray[nodeCount].m_LeftChildIndx=self.m_CurrNodeNum
                    self.m_NodeArray[nodeCount].m_RightChildIndx=self.m_CurrNodeNum+1
                    lci = self.m_CurrNodeNum
                    rci = self.m_CurrNodeNum+1

                    self.m_NodeArray[lci].setNode(lci,self.m_NodeArray[nodeCount].m_NodeDepth+1,
                            self.m_NodeArray[nodeCount].m_NodeIndx)

                    self.m_NodeArray[rci].setNode(rci,self.m_NodeArray[nodeCount].m_NodeDepth+1,
                            self.m_NodeArray[nodeCount].m_NodeIndx)

                    self.splitDataFile(self.m_NodeArray[nodeCount],data_left,data_right)

                    self.m_CurrNodeNum = self.m_CurrNodeNum+2

                self.printNodeData(self.m_NodeArray[nodeCount])

            else:
                print("Tree Model Trained!!!!!!!!")
                break
```

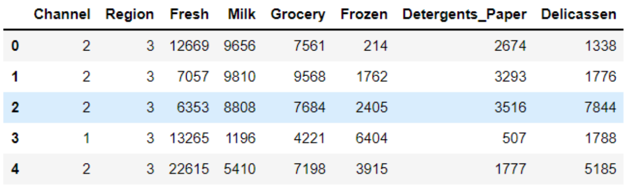
10. Loading the data and performing Hierarchical Kmeans-



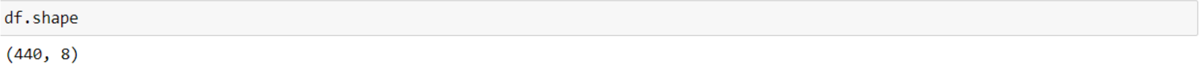
```
df = pd.read_csv("Wholesale customers data.csv")
```



```
df.head()
```

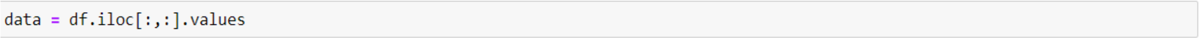


	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

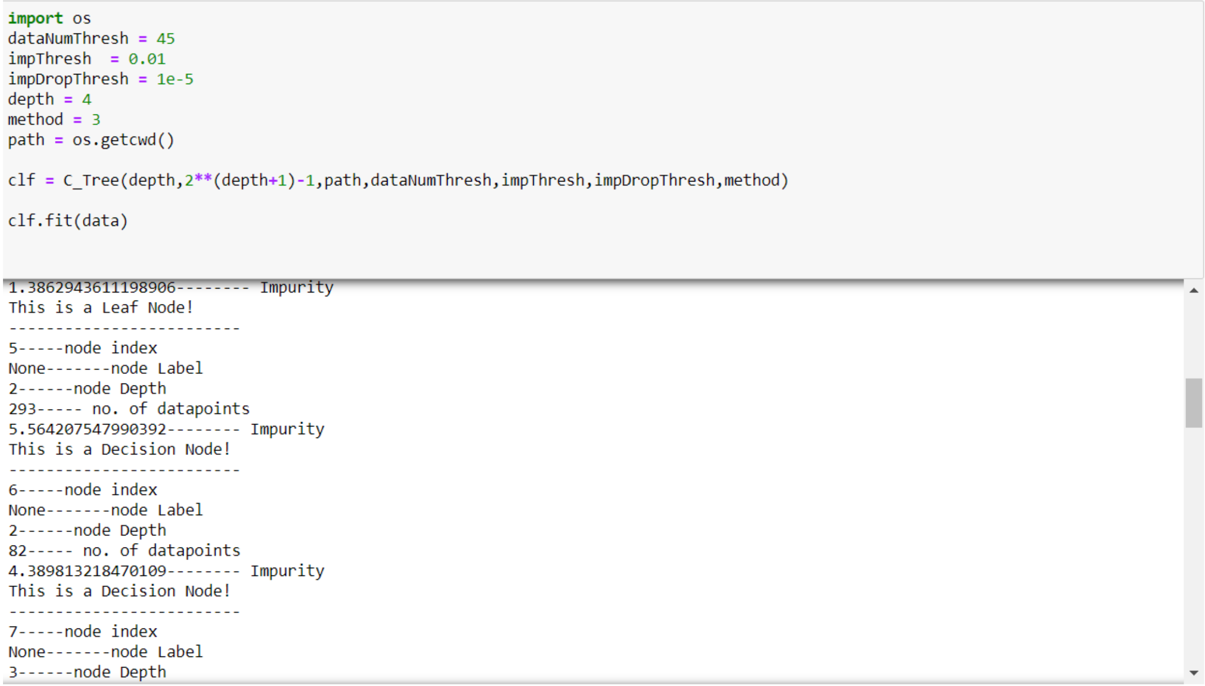


```
df.shape
```

```
(440, 8)
```



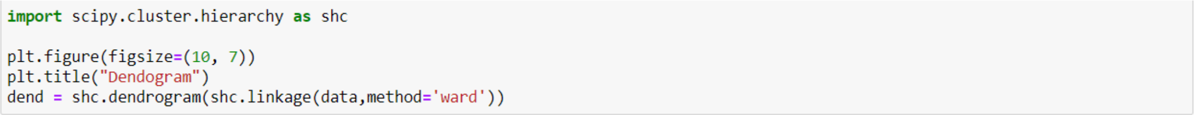
```
data = df.iloc[:,:].values
```



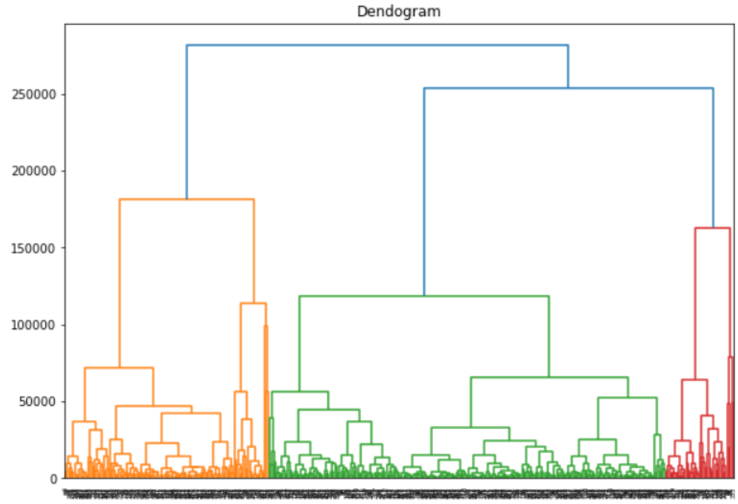
```
import os
dataNumThresh = 45
impThresh  = 0.01
impDropThresh = 1e-5
depth = 4
method = 3
path = os.getcwd()

clf = C_Tree(depth,2**(depth+1)-1,path,dataNumThresh,impThresh,impDropThresh,method)

clf.fit(data)
```

```
1.3862943611198906-------- Impurity
This is a Leaf Node!
-------------------------
5-----node index
None-------node Label
2------node Depth
293----- no. of datapoints
5.5642075479903392-------- Impurity
This is a Decision Node!
-------------------------
6-----node index
None-------node Label
2------node Depth
82----- no. of datapoints
4.389813218470109-------- Impurity
This is a Decision Node!
-------------------------
7-----node index
None-------node Label
3------node Depth
```

11. Hierarchical Clustering using sklearn-



```
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))
plt.title("Dendogram")
dend = shc.dendrogram(shc.linkage(data,method='ward'))
```



```
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
cluster.fit_predict(data)
```

```
array([0, 0, 0, 0, 3, 0, 0, 0, 0, 4, 0, 0, 3, 3, 3, 0, 0, 0, 3, 0, 3, 0,
       3, 0, 3, 3, 0, 3, 4, 1, 3, 0, 3, 3, 0, 0, 3, 0, 4, 1, 3, 3, 0, 4,
       0, 4, 4, 2, 0, 4, 0, 0, 1, 0, 3, 0, 4, 0, 3, 0, 0, 2, 0, 0, 0, 4,
       0, 3, 0, 0, 3, 3, 0, 3, 0, 3, 0, 4, 0, 0, 0, 0, 0, 3, 0, 2, 2, 1,
       0, 3, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 3, 0, 0, 0, 4,
       0, 0, 3, 3, 3, 0, 0, 0, 3, 0, 3, 0, 3, 0, 1, 1, 3, 3, 0, 1, 0, 0,
       3, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 0, 3, 4, 0, 0, 0, 3, 3, 0, 3, 0,
       0, 4, 0, 3, 0, 0, 0, 3, 3, 4, 0, 0, 0, 0, 0, 0, 0, 4, 0, 4, 0, 0,
       1, 0, 0, 0, 3, 1, 0, 1, 0, 0, 0, 0, 0, 0, 3, 3, 0, 4, 0, 3, 3, 0,
       0, 0, 4, 4, 3, 0, 0, 4, 0, 0, 0, 4, 3, 4, 0, 0, 0, 0, 4, 3, 0, 0,
       3, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 3, 0, 1, 3, 3,
       3, 0, 0, 0, 0, 3, 3, 0, 0, 4, 0, 3, 0, 3, 0, 0, 1, 1, 0, 0, 3, 0,
       0, 0, 4, 3, 0, 3, 0, 0, 0, 1, 0, 0, 3, 0, 0, 3, 0, 0, 1, 3, 1, 1,
       0, 3, 3, 1, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 3, 4, 0, 0, 4, 0, 4, 3,
       0, 0, 0, 3, 4, 0, 0, 0, 0, 0, 0, 4, 0, 0, 3, 3, 3, 1, 0, 0, 3, 0,
       0, 4, 3, 2, 3, 3, 3, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 3, 0, 4, 0, 4,
       0, 4, 0, 0, 3, 4, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 3, 0, 1, 3, 0, 3,
       0, 0, 0, 1, 0, 0, 3, 3, 3, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 0,
       0, 0, 0, 0, 0, 3, 3, 3, 3, 0, 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       4, 0, 0, 3, 3, 3, 3, 0, 0, 3, 0, 0, 0, 0, 3, 0, 3, 3, 1, 4, 0, 0],
      dtype=int64)
```