DATA PRE PROCESSING AND HEIRARCHICAL CLUSTERING (ASSESSMENT - 5)

CSE4020(MACHINE LEARNING)LAB:L49-L50

APRIL 24, 2022

ANIRUDH VADERA
20BCE2940

QUESTION:

Q1. Classify the model based on the process listed below.

1. **Data Pre-processing**
    - Fill the missing values using any imputation method
    - Normalize the data
    - Display the data set
2. **Feature Selection**
    - Select the best feature set using Principal Component Analysis and display the top 20% features.
3. **Handling imbalance data**
    - Find out the percentage of data in each class of the training sample. If found imbalance, apply any one of the balancing technique to make it as balanced data.
4. **Model Fit**
    - Train the data prepared in earlier step using random forest model and display the evaluation metrics for the model.
5. **Evaluation Metrics and Visualization**
    - Display accuracy, precision, recall and F1 Score.
    - Display accuracy, precision, recall and F1 Score.

Q2. Hierarchical clustering

Perform hierarchical clustering using ward linkage method for the chosen data set. Display the cluster in the form of Dendrogram and apply suitable metric to find the optimal cluster. Also display the Sum of Squared Error (SSE) value.

Q1: Description:

Data Pre-processing:

1. **For filling in the missing values we have used the SimpleImputer from sklearn**

```python
from sklearn.impute import SimpleImputer
imputer=SimpleImputer()
imputedData=pd.DataFrame(imputer.fit_transform(df))
print("number of nan values after imputation :",imputedData.isna().sum().sum())
```

Python

2. **For normalizing the data we have used MinMaxScaller**

```python
scaler = preprocessing.MinMaxScaler()
names = df.columns
df_normalized = scaler.fit_transform(imputedData)
scaled_df = pd.DataFrame(df_normalized, columns=names)
scaled_df.head()
```

Feature Selection:

We have used PCA function with 20% i.e 4 top attributes

```python
pca_20=PCA(n_components=int(22*0.2))
pca_20.fit(scaled_df)
df_pca_20=pca_20.transform(scaled_df)
df_pca_20.shape
pd.DataFrame(df_pca_20).head()
```

Handling Imbalance Data:

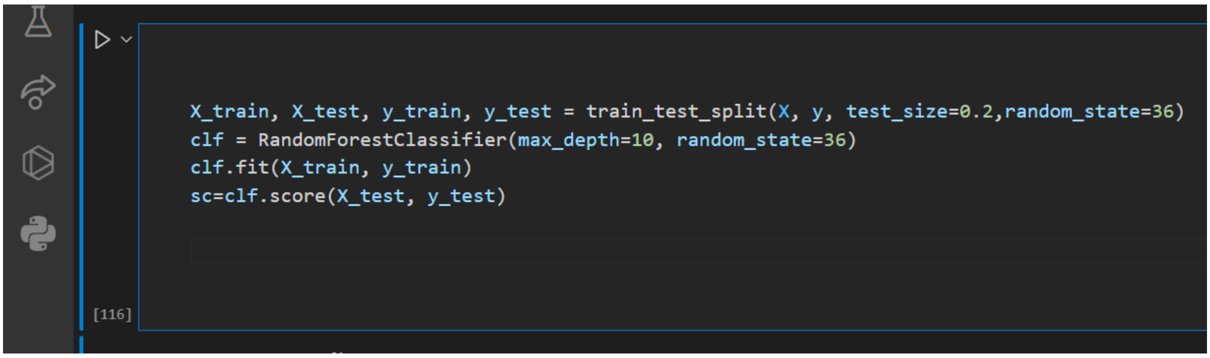
We have used RandomOverSampler

```python
ros = RandomOverSampler()
print(utils.multiclass.type_of_target(y.astype('int')))
X, y = ros.fit_resample(df_pca_20, y.astype('int'))
sm = SMOTE(random_state=0,k_neighbors=3)
X, y = sm.fit_resample(X, y)
```

[107]

Model Fit:

Using Random Forest:



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=36)
clf = RandomForestClassifier(max_depth=10, random_state=36)
clf.fit(X_train, y_train)
sc=clf.score(X_test, y_test)
```

[116]

Code:

```
import pandas as pd

import numpy as np

from sklearn import preprocessing

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

from imblearn.over_sampling import RandomOverSampler

from imblearn.over_sampling import SMOTE


# from sklearn import utils

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.impute import SimpleImputer

from sklearn.metrics import mean_absolute_error,mean_squared_error,f1_score,recall_score

from sklearn.metrics import plot_confusion_matrix, confusion_matrix, accuracy_score, precision_score, recall_score,classification_report



df=pd.read_csv("C:/Users/Anirudh/OneDrive/Desktop/kamyr-digester.csv")

df=df.drop(["Observation"],axis=1)
```

```
y=df["Y-Kappa"]
df=df.drop(["Y-Kappa"],axis=1)

print("ANIRUDH VADERA (20BCE2940)")

print("The df is as following : ")
print(df)
print("\n")

# Check for missing values
print("Checking for missing values :")
print(df.isnull().sum())
print("\n")

# Check for NAN values
print("Number of nan values before imputations : ")
print(df.isna().sum().sum())
print("\n")

# Printing the header of the df
print("df Header : ")
print(df.head())
print("\n")

# Information regarding the columns
print("Information regarding the columns : ")
print(df.info())
print("\n")
```

```
# Information related to the df
print("df Details : ")
print(df.describe())
print("\n")

imputer=SimpleImputer()
imputedData=pd.DataFrame(imputer.fit_transform(df))
print("Number of NAN values after Simple Imputation :",imputedData.isna().sum().sum())

scaler = preprocessing.MinMaxScaler()
names = df.columns
df_normalized = scaler.fit_transform(imputedData)
scaled_df = pd.DataFrame(df_normalized, columns=names)
scaled_df.head()

pca_20=PCA(n_components=int(22*0.2))
pca_20.fit(scaled_df)
df_pca_20=pca_20.transform(scaled_df)
df_pca_20.shape
pd.DataFrame(df_pca_20).head()

x=[]
for i in range(len(y)):
   x.append(i)
plt.hist(y)
for i in (plt.hist(y)[1] ) :
   print("Percentage of Data : " ,(i/sum(plt.hist(y)[1]))*100,"%")
plt.show()
```

```
print(df_pca_20.shape)


ros = RandomOverSampler()
X, y = ros.fit_resample(df_pca_20, y.astype('int'))
sm = SMOTE(random_state=0,k_neighbors=3)
X, y = sm.fit_resample(X, y)


x=[]
for i in range(len(y)):
    x.append(i)
plt.hist(y)
for i in (plt.hist(y)[1] ) :
    print("Percentage of Data : " ,(i/sum(plt.hist(y)[1]))*100,"%")
plt.show()


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=36)
clf = RandomForestClassifier(max_depth=10, random_state=36)
clf.fit(X_train, y_train)
y_pred=clf.predict(X_test)
sc=clf.score(X_test, y_test)


# Checking the accuracy of our model
print('Accuracy: ',accuracy_score(y_test,y_pred)*100,"%")
print('Precision: %.3f' % precision_score(y_test, y_pred,average='micro'))
print('Recall: %.3f' % recall_score(y_test, y_pred,average='micro'))
print("Mean Absolute Error:",mean_absolute_error(y_test,y_pred).round(2))
print("Mean Squared Error:",mean_squared_error(y_test,y_pred).round(2))
```

```
# Our Model Report
print('*************** Evaluation on Our Model ***************')
score_te = clf.score(X_test, y_test)
print('Accuracy Score: ', score_te)
# Look at classification report to evaluate the model
print(classification_report(y_test, y_pred))
print('--------------------------------------------------------')
print("")


# Additional Plots
import seaborn as sns
sns.pairplot(pd.concat([pd.DataFrame(X),pd.DataFrame(y)],axis=1))
```

Output and Results:

Data Pre-processing:

Dataset:

```
ANIRUDH VADERA (20BCE2940)
The df is as following :
     ChipRate  BF-CMratio  ...  T-Top-Chips-4  SulphidityL-4
0      16.520     121.717  ...        252.077            NaN
1      16.810      79.022  ...        251.406          29.11
2      16.709      79.562  ...        251.335            NaN
3      16.478      81.011  ...        250.312          29.02
4      15.618      93.244  ...        249.916          29.01
..        ...         ...  ...            ...            ...
296    14.233      89.790  ...        252.311            NaN
297    15.167      84.640  ...        251.833          30.29
298       NaN      85.034  ...        251.614          30.47
299       NaN      88.013  ...        251.197            NaN
300       NaN      85.490  ...        251.324          30.46

[301 rows x 21 columns]
```

```
Checking for missing values :
ChipRate            4
BF-CMratio         14
BlowFlow           13
ChipLevel4          1
T-upperExt-2        1
T-lowerExt-2        1
UCZAA              24
WhiteFlow-4         1
AAWhiteSt-4       141
AA-Wood-4           1
ChipMoisture-4      1
SteamFlow-4         1
Lower-HeatT-3       1
Upper-HeatT-3       1
ChipMass-4          1
WeakLiquorF         1
BlackFlow-2         1
WeakWashF           1
SteamHeatF-3        1
T-Top-Chips-4       1
SulphidityL-4     141
dtype: int64
```

The missing values are not null:

```
In [6]:
   ...: print("Number of nan values before imputations : ")
   ...: print(df.isna().sum().sum())
   ...: print("\n")
Number of nan values before imputations :
352
```

After imputation(SimpleImputation) the Nan Values are none.

```
In [5]: imputer=SimpleImputer()
   ...: imputedData=pd.DataFrame(imputer.fit_transform(df))
   ...: print("Number of NAN values after Simple Imputation :",imputedData.isna().sum().sum())
Number of NAN values after Simple Imputation : 0
```

Displaying the Dataset after imputation (Dataset Details):

```
df Header :
        0        1         2        3  ...       17      18       19         20
0  16.520  121.717  1177.607  169.805  ...  257.325  54.612  252.077  30.463594
1  16.810   79.022  1328.360  341.327  ...  241.182  46.603  251.406  29.110000
2  16.709   79.562  1329.407  239.161  ...  237.272  51.795  251.335  30.463594
3  16.478   81.011  1334.877  213.527  ...  239.478  54.846  250.312  29.020000
4  15.618   93.244  1334.168  243.131  ...  215.372  54.186  249.916  29.010000

[5 rows x 21 columns]


            0        1         2        3  ...       17      18       19         20
0    16.52000  121.717  1177.607  169.805  ...  257.325  54.612  252.077  30.463594
1    16.81000   79.022  1328.360  341.327  ...  241.182  46.603  251.406  29.110000
2    16.70900   79.562  1329.407  239.161  ...  237.272  51.795  251.335  30.463594
3    16.47800   81.011  1334.877  213.527  ...  239.478  54.846  250.312  29.020000
4    15.61800   93.244  1334.168  243.131  ...  215.372  54.186  249.916  29.010000
..        ...      ...       ...      ...  ...      ...     ...      ...        ...
296  14.23300   89.790  1278.006  379.458  ...  388.676  47.803  252.311  30.463594
297  15.16700   84.640  1283.706  339.440  ...  388.911  49.524  251.833  30.290000
298  14.33867   85.034  1278.345  368.564  ...  418.979  48.135  251.614  30.470000
299  14.33867   88.013  1307.722  278.842  ...  462.712  54.373  251.197  30.463594
300  14.33867   85.490  1255.986  273.484  ...  457.313  53.194  251.324  30.460000

[301 rows x 21 columns]
```

After Scaling(MinMaxScaller) the dataset looks like:

```
   ChipRate  BF-CMratio  ...  T-Top-Chips-4  SulphidityL-4
0  0.937204    1.000000  ...       0.645150       0.379528
1  0.978781    0.195527  ...       0.528718       0.026110
2  0.964301    0.205702  ...       0.516398       0.379528
3  0.931183    0.233004  ...       0.338886       0.002611
4  0.807885    0.463502  ...       0.270172       0.000000

[5 rows x 21 columns]
```

```
     ChipRate  BF-CMratio  ...  T-Top-Chips-4  SulphidityL-4
0    0.937204    1.000000  ...       0.645150       0.379528
1    0.978781    0.195527  ...       0.528718       0.026110
2    0.964301    0.205702  ...       0.516398       0.379528
3    0.931183    0.233004  ...       0.338886       0.002611
4    0.807885    0.463502  ...       0.270172       0.000000
..        ...         ...  ...            ...            ...
296  0.609319    0.398421  ...       0.685754       0.379528
297  0.743226    0.301383  ...       0.602811       0.334204
298  0.624469    0.308807  ...       0.564810       0.381201
299  0.624469    0.364938  ...       0.492452       0.379528
300  0.624469    0.317399  ...       0.514489       0.378590

[301 rows x 21 columns]
```

Feature Selection:

We have used PCA function with 20% i.e 4 top attributes

```
          0         1         2         3
0  0.389993 -0.469850 -0.177754  0.105990
1  0.654146 -0.081976  0.423638  0.298261
2  0.523610 -0.233978  0.384205  0.274731
3  0.252892 -0.585233  0.270943  0.354704
4  0.164492 -0.523645  0.014838  0.316388

In [17]: df_pca_20
Out[17]:
array([[ 0.38999265, -0.46984995, -0.17775449,  0.10598979],
       [ 0.65414624, -0.08197591,  0.42363755,  0.29826133],
       [ 0.52361007, -0.23397812,  0.38420465,  0.27473115],
       ...,
       [ 0.71662265, -0.0978874 ,  0.01524913, -0.29802021],
       [ 0.5665067 , -0.30132541,  0.06296998, -0.13262049],
       [ 0.55944449, -0.52447543, -0.06957225, -0.24557584]])
```

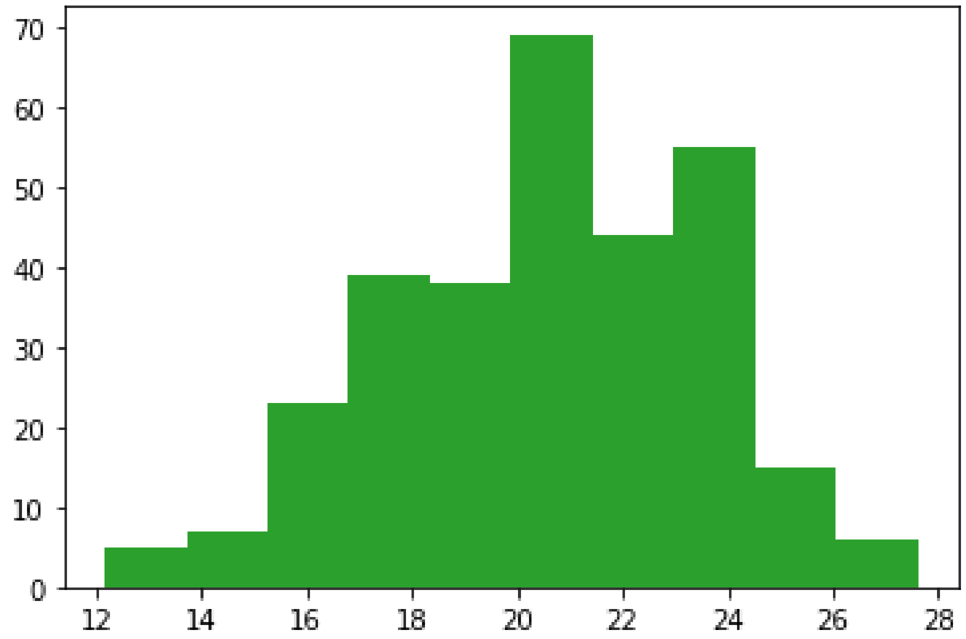
Handling Imbalance data:

Percentage of data

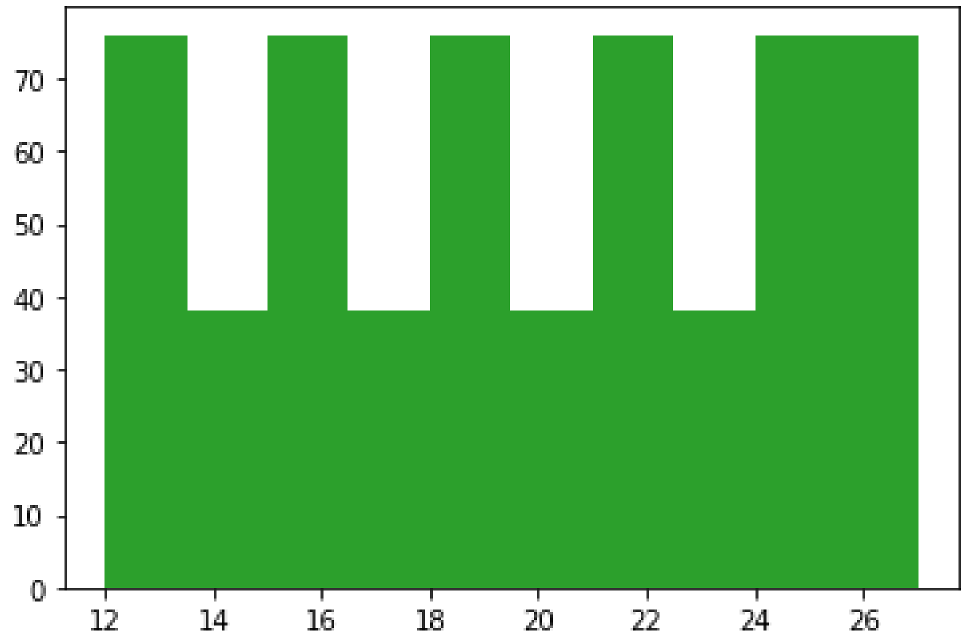
```
Percentage of Data :  5.563810089834 73 %
```

```
Percentage of Data :  5.56381008983473 %
Percentage of Data :  6.269229890049603 %
Percentage of Data :  6.974649690264474 %
Percentage of Data :  7.680069490479345 %
Percentage of Data :  8.385489290694217 %
Percentage of Data :  9.09090909090909 %
Percentage of Data :  9.796328891123961 %
Percentage of Data :  10.501748691338834 %
Percentage of Data :  11.207168491553706 %
Percentage of Data :  11.912588291768579 %
Percentage of Data :  12.618008091983448 %
```

The distribution of data initially(Imbalanced):



The distribution of data after using RandomOverSampler(balanced):



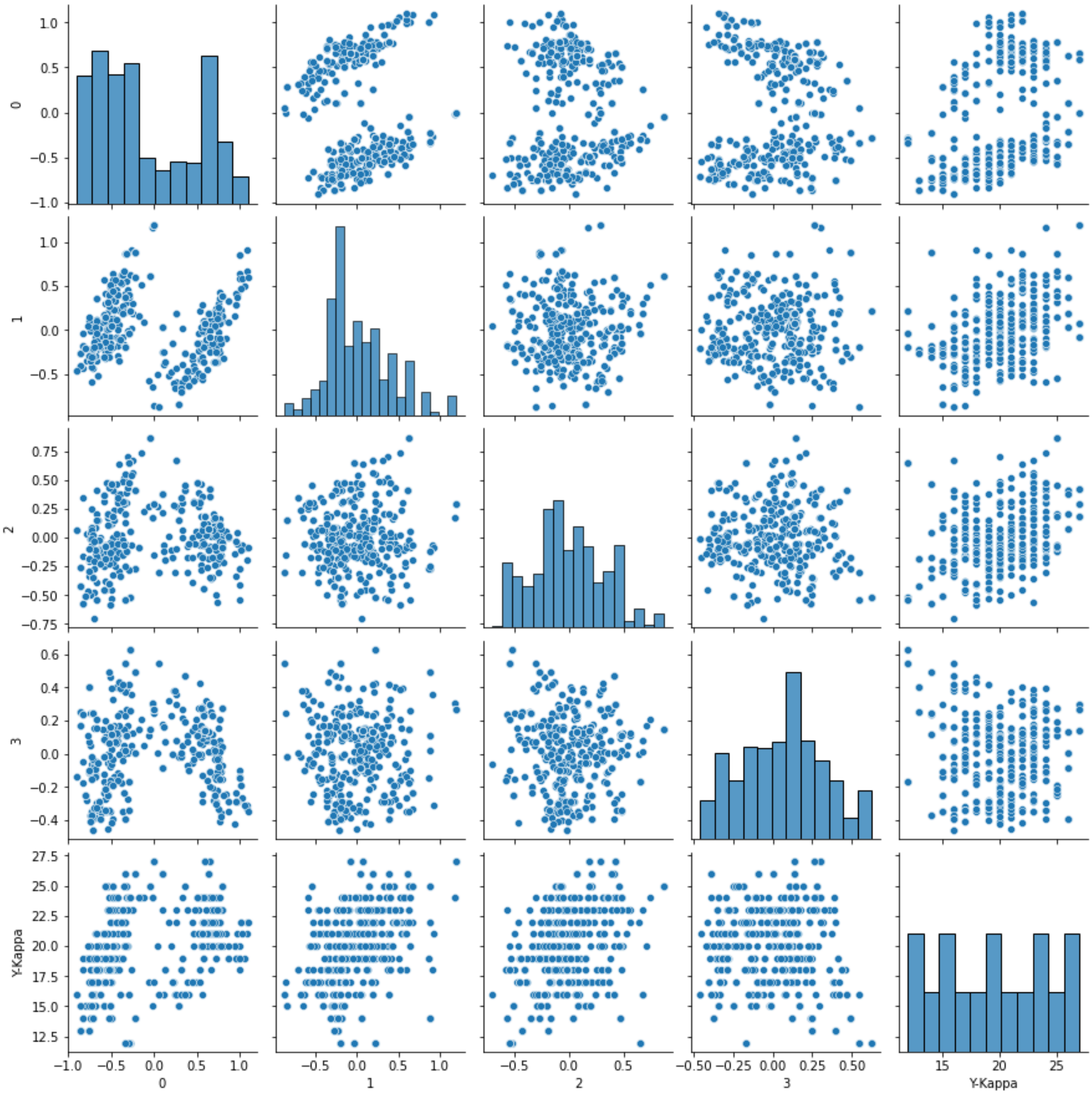
Accuracy analysis(RandomForest model):

```
Accuracy:  91.67213114754098 %
Precision: 0.697
Recall: 0.697
Mean Absolute Error: 0.75
Mean Squared Error: 2.7
```

```
*************** Evaluation on Our Model ***************
Accuracy Score:  91.67213114754098
              precision    recall  f1-score   support

          12       1.00      1.00      1.00         9
          13       1.00      1.00      1.00        10
          14       1.00      1.00      1.00        10
          15       0.67      1.00      0.80         8
          16       0.67      0.60      0.63        10
          17       0.75      0.43      0.55         7
          18       0.80      0.57      0.67         7
          19       0.33      0.20      0.25         5
          20       0.50      0.33      0.40         6
          21       0.00      0.00      0.00         9
          22       0.22      0.50      0.31         4
          23       0.00      0.00      0.00         4
          24       0.50      0.62      0.56         8
          25       0.92      1.00      0.96        11
          26       1.00      1.00      1.00         6
          27       1.00      1.00      1.00         8
```

Pairplot between different features:



Inference:

- **All the not applicable values were removed from dataset successfully**
- **The dataset was balanced to a good extent as the accuracy received is 91%**

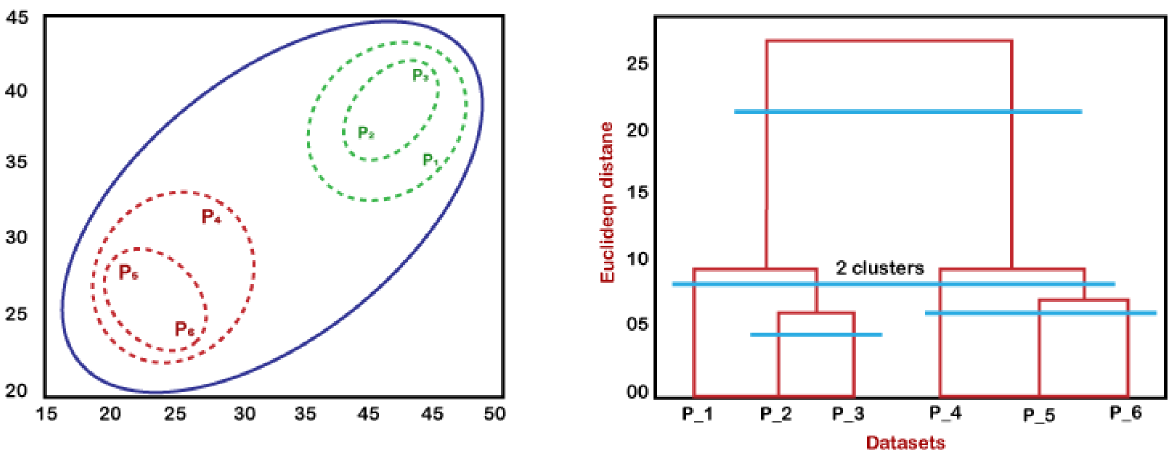
Q2: (Hierarchical Clustering)

Description:

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**.

The working of the dendrogram can be explained using the below diagram:



In the above diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.

- Firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The hight is decided according to the Euclidean distance between the data points.
- In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.
- Again, two new dendrograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
- At last, the final dendrogram is created that combines all the data points together.

We can cut the dendrogram tree structure at any level as per our requirement.

Code:

```
# Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
# Importing the dataset
dataset =
pd.read_csv('C:/Users/Anirudh/OneDrive/Desktop/Mall_Customers.csv')
print("ANIRUDH VADERA (20BCE2940)")
print("The dataset is as following : ")
print(dataset)
print("\n")
# Check for missing values
print("Checking for missing values :")
print(dataset.isnull().sum())
print("\n")
# Printing the header of the dataset
print("Dataset Header : ")
print(dataset.head())
print("\n")
# Information regarding the columns
print("Information regarding the columns : ")
print(dataset.info())
print("\n")
# Information related to the dataset
print("Dataset Details : ")
```

```
print(dataset.describe())
print("\n")
# Choosing the variabled that is of our use
x = dataset.iloc[:, [3, 4]].values
col1 = dataset.iloc[:,3].values
col2 = dataset.iloc[:,4].values
y_test = []
for i in range(len(dataset)):
    if(col1[i]<=60):
        y_test.append(0)
    elif(col1[i]<=130):
        if(col1[i]<=130 and col2[i]<=100 and col2[i]>60):
            y_test.append(4)
        else:
            y_test.append(3)
    elif(col1[i]<=220 and col2[i]<=80):
        y_test.append(2)
    elif(col1[i]<=300 and col2[i]<=100):
        y_test.append(1)
    else:
        y_test.append(1)


print(y_test)


dataset["Prediction"] = y_test
```

```
dataset.Prediction=dataset.Prediction.replace({0:"low income and mid spending", 1:"high income and high spending", 2:"mid income and mid spending", 3:"low income and low spending", 4:"low income and high spending"})



#Finding the optimal number of clusters using the dendrogram

import scipy.cluster.hierarchy as shc

mtp.figure(figsize=(18, 50))

dendro = shc.dendrogram(shc.linkage(x, method="ward"),leaf_rotation=0, leaf_font_size=12, orientation='right')

mtp.title("Dendrogrma Plot")

mtp.ylabel("Euclidean Distances")

mtp.xlabel("Customers")

mtp.show()


#training the hierarchical model on dataset

from sklearn.cluster import AgglomerativeClustering

hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')

y_pred= hc.fit_predict(x)


mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')

mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')

mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
```

mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

mtp.title('Clusters of customers')

mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend(loc='upper left')

mtp.show()

print("Sum of squared error: %.2f" % (sum(pow(y_pred-y_test,2))))

Output and Results:

Data Pre-processing:

Dataset:

```
ANIRUDH VADERA (20BCE2940)
The dataset is as following :
     CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          ...     ...  ...                 ...                     ...
245         246    Male   30                 297                      69
246         247  Female   56                 311                      14
247         248    Male   29                 313                      90
248         249  Female   19                 316                      32
249         250  Female   31                 325                      86

[250 rows x 5 columns]


Checking for missing values :
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

There are no missing values therefore we can move forward:

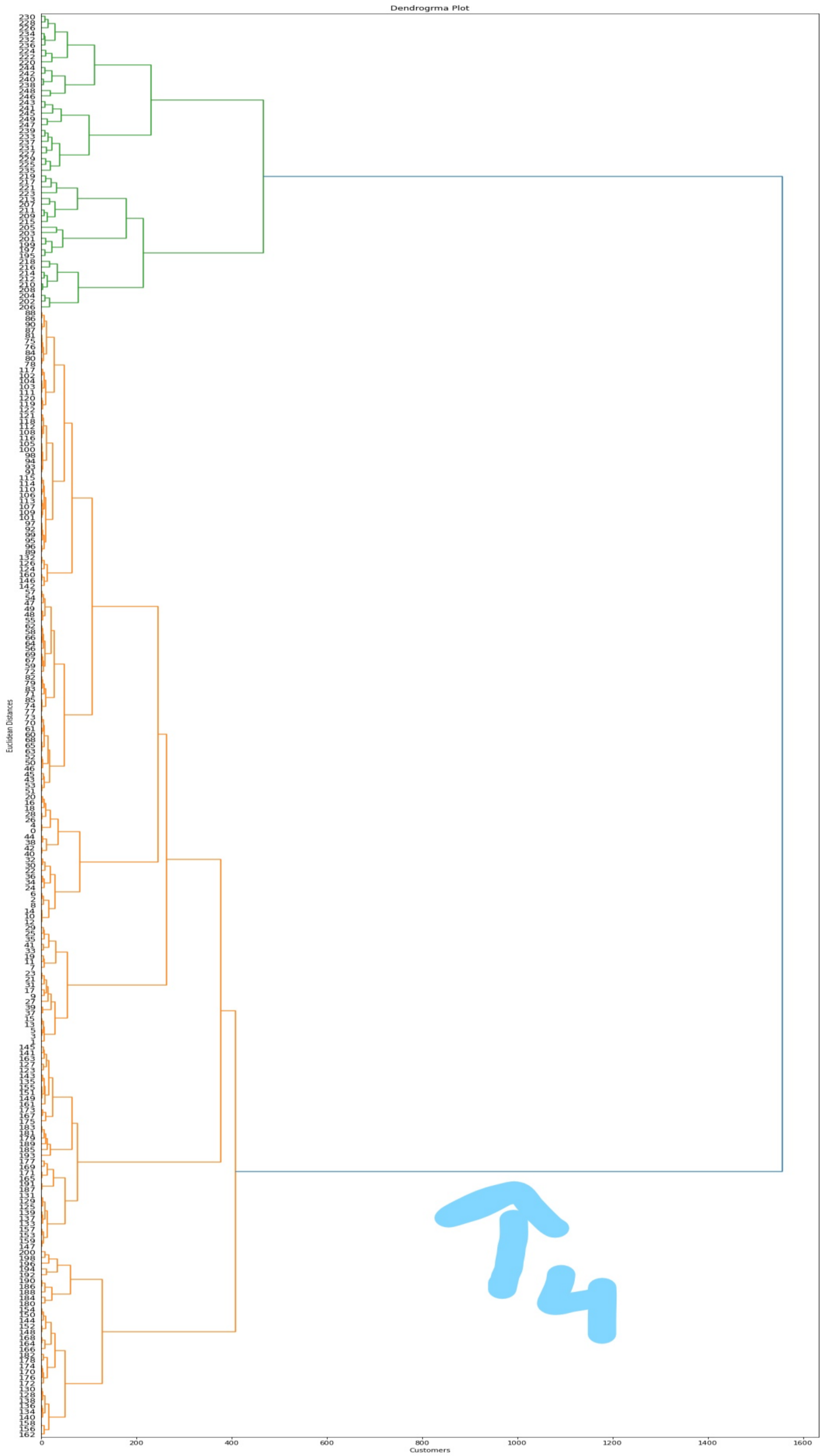
Dataset Details:

```
Dataset Header :
   CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40


Information regarding the columns :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              250 non-null    int64
 1   Gender                  250 non-null    object
 2   Age                     250 non-null    int64
 3   Annual Income (k$)      250 non-null    int64
 4   Spending Score (1-100)  250 non-null    int64
dtypes: int64(4), object(1)
memory usage: 9.9+ KB
None
```

```
Dataset Details :
       CustomerID        Age  Annual Income (k$)  Spending Score (1-100)
count  250.000000  250.00000          250.000000              250.000000
mean   125.500000   38.49200           95.592000               50.244000
std     72.312977   13.17026           77.308758               27.289914
min      1.000000   18.00000           15.000000                1.000000
25%     63.250000   29.00000           47.000000               27.000000
50%    125.500000   36.00000           70.000000               50.000000
75%    187.750000   47.75000          101.000000               74.000000
max    250.000000   70.00000          325.000000               99.000000
```

Finding the optimal number of clusters using the Dendrogram:


Dendrogrma Plot
Euclidean Distances
Customers
0
200
400
600
800
1000
1200
1400
1600
4

Used linkage method is ward.

Using this Dendrogram, we will now determine the optimal number of clusters for our model. For this, we will find the **maximum vertical distance** that does not cut any horizontal bar.

As we can visualize, the 4th distance is looking the maximum, so according to this, **the number of clusters will be 5**(the vertical lines in this range)

So, the optimal number of clusters will be 5, and we will train the model in the next step, using the same.

Training the hierarchical clustering model:

Parameters:

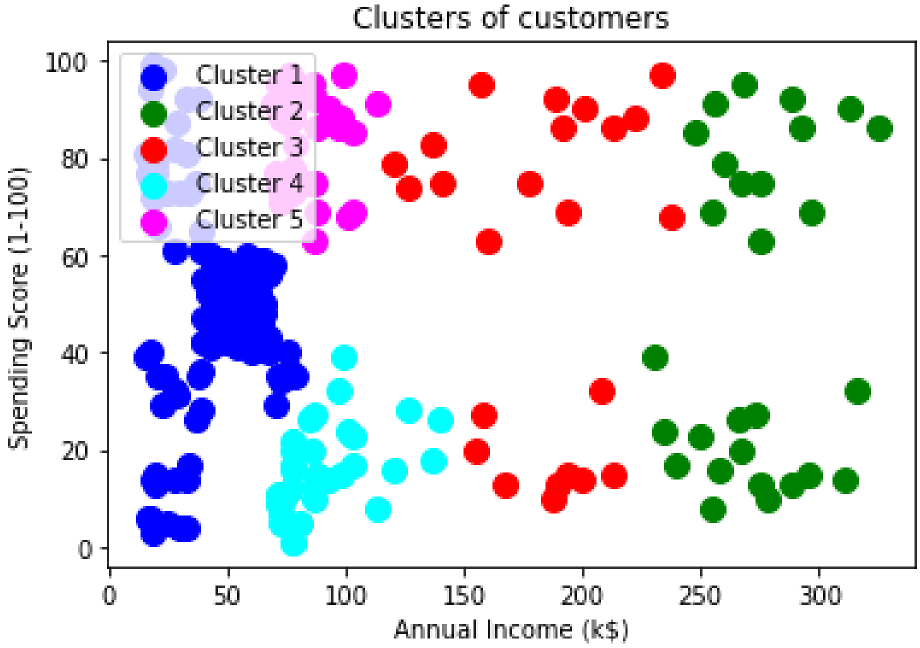
- **n_clusters=5**
- **affinity='euclidean'**: It is a metric used to compute the linkage.
- **linkage='ward'**

The Y_test is as follows:

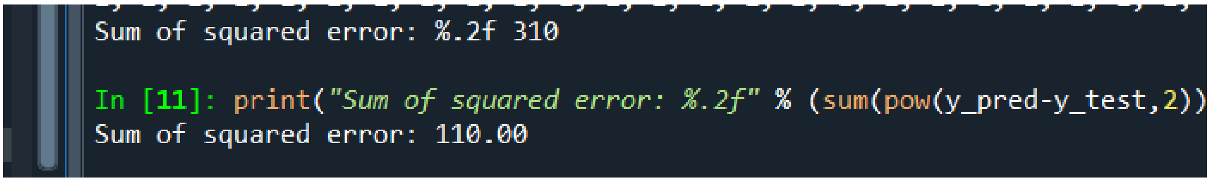
```
In [18]: dataset['Prediction']
Out[18]:
0        low income and mid spending
1        low income and mid spending
2        low income and mid spending
3        low income and mid spending
4        low income and mid spending
                     ...
245    high income and high spending
246    high income and high spending
247    high income and high spending
248    high income and high spending
249    high income and high spending
Name: Prediction, Length: 250, dtype: object
```

Visualizing the clusters:

- Cluster 1 is low income and mid spending
- Cluster 2 is high income and high spending
- Cluster 3 is mid income and mid spending
- Cluster 4 is low income and low spending
- Cluster 5 is low income and high spending



SSE(Sum of Squared Errors:)



Inference:

- Cluster 1 is low income and mid spending
- Cluster 2 is high income and high spending
- Cluster 3 is mid income and mid spending
- Cluster 4 is low income and low spending
- Cluster 5 is low income and high spending
- Using the maximum vertical distance we find the optimum number of clusters