

**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

1. **Feature Selection**

* Select the best feature set using Principal Component Analysis and display the top 20% features.

1. **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.

1. **Model Fit**

* Train the data prepared in earlier step using random forest model and display the evaluation metrics for the model.

1. **Evaluation Metrics and Visualization**

* Display accuracy, precision, recall and F1 Score.
* Display accuracy, precision, recall and F1 Score.

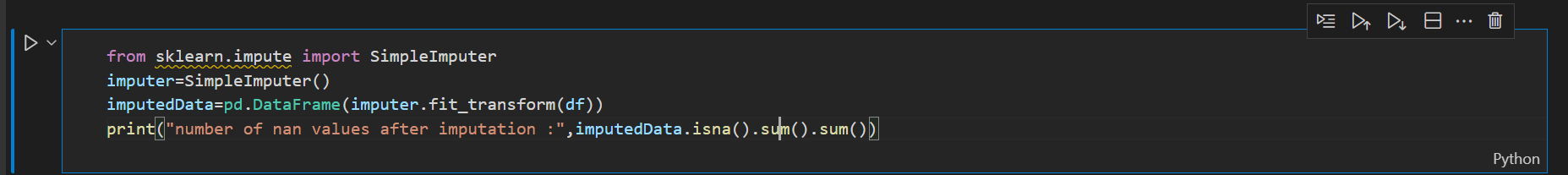
**Q2. Hierarchical clustering**

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.

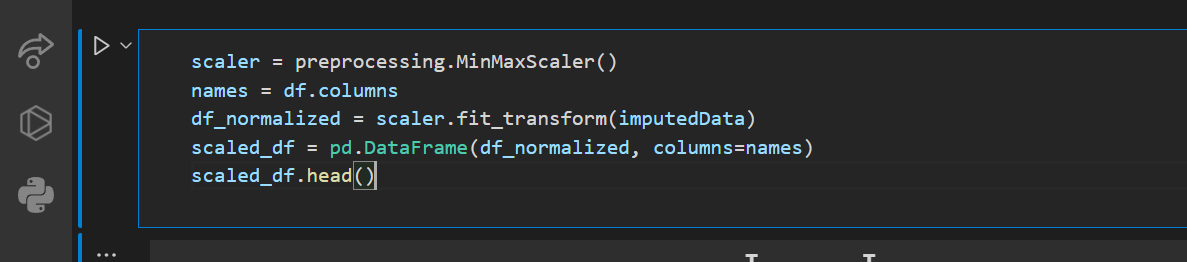
**Q1: Description:**

**Data Pre-processing:**

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

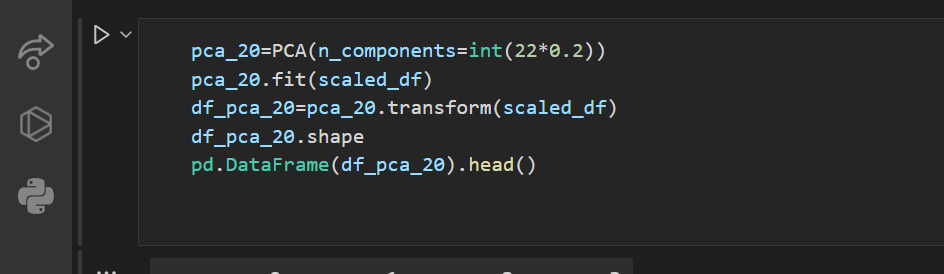
****

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

****

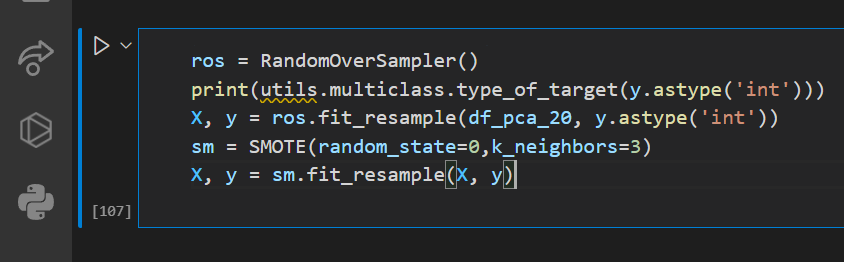
**Feature Selection:**

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

****

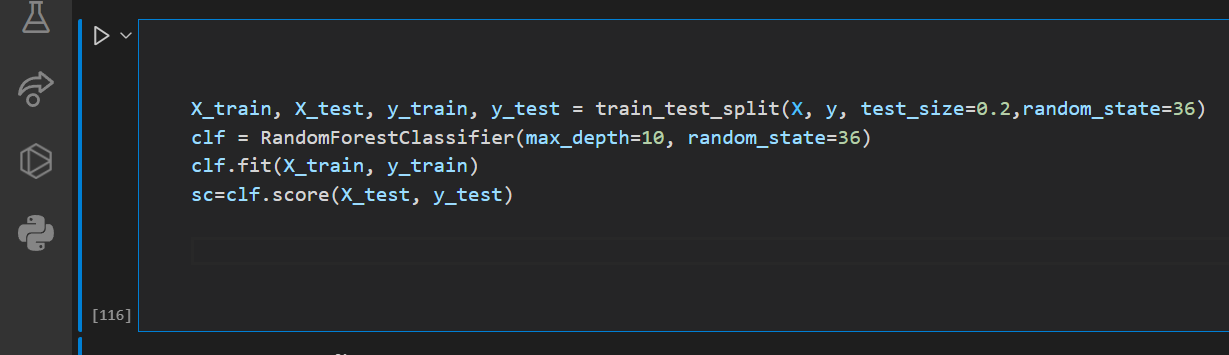
**Handling Imbalance Data:**

**We have used RandomOverSampler**

****

**Model Fit:**

**Using Random Forest:**

****

**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

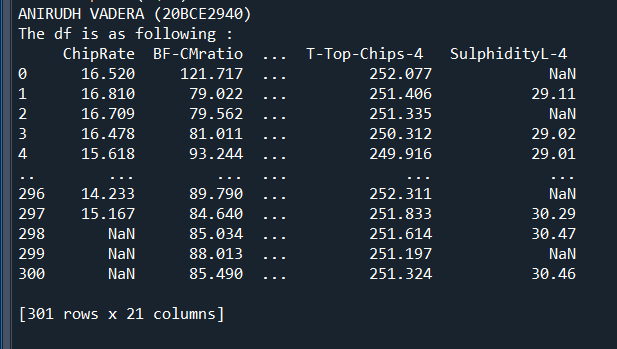
import seaborn as sns

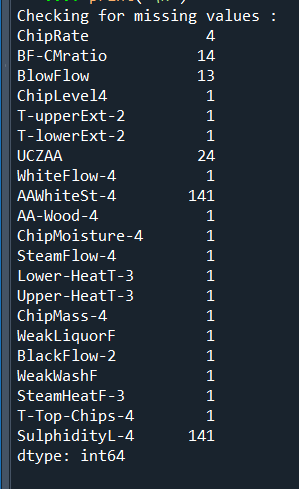
sns.pairplot(pd.concat([pd.DataFrame(X),pd.DataFrame(y)],axis=1))

**Output and Results:**

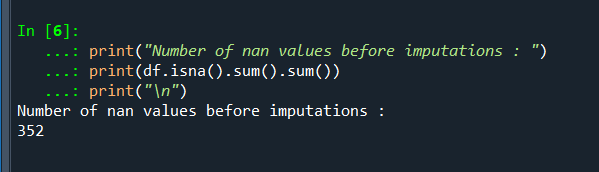
**Data Pre-processing:**

**Dataset:**

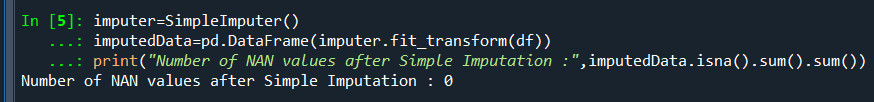
****

****

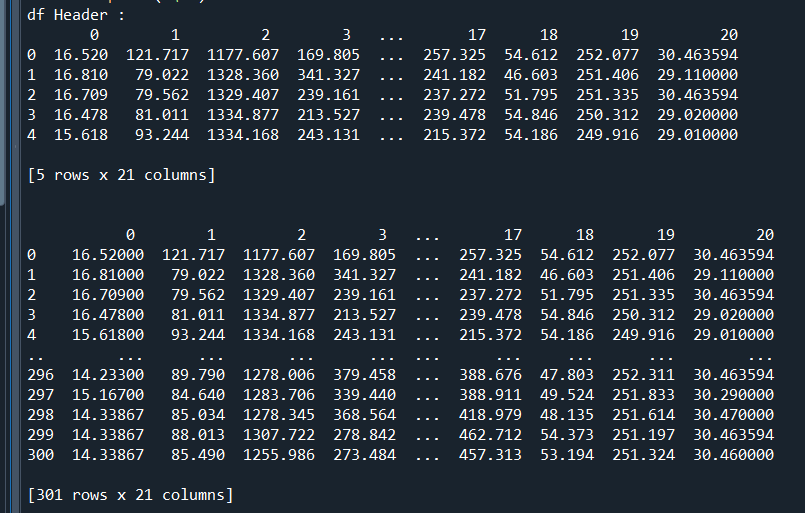
**The missing values are not null:**

****

**After imputation(SimpleImputation) the Nan Values are none.**

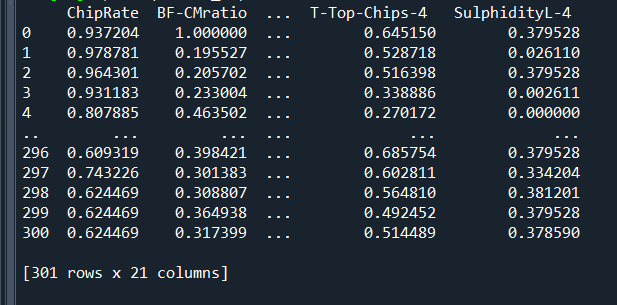
****

**Displaying the Dataset after imputation (Dataset Details):**

****

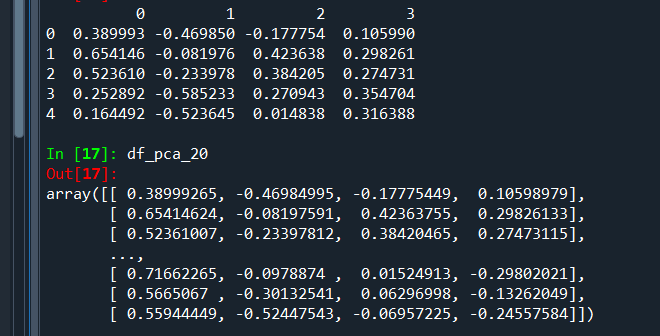
**After Scaling(MinMaxScaller) the dataset looks like:**

****

****

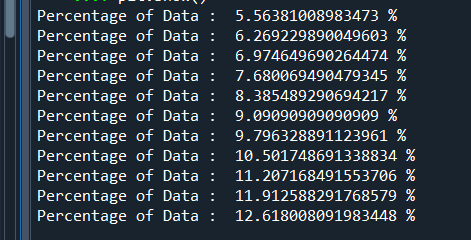
**Feature Selection:**

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

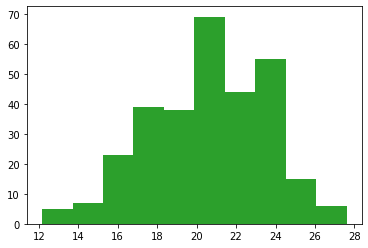
****

**Handling Imbalance data:**

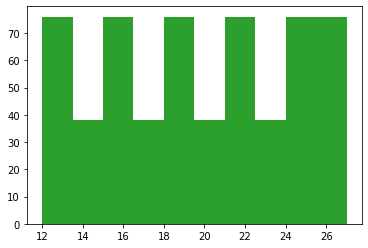
**Percentage of data**

****

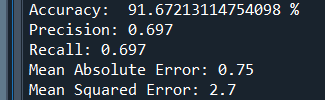
**The distribution of data initially(Imbalanced):**

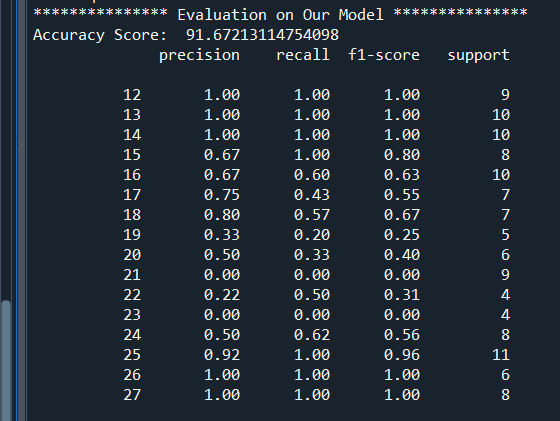


**The distribution of data after using RandomOverSampler(balanced):**

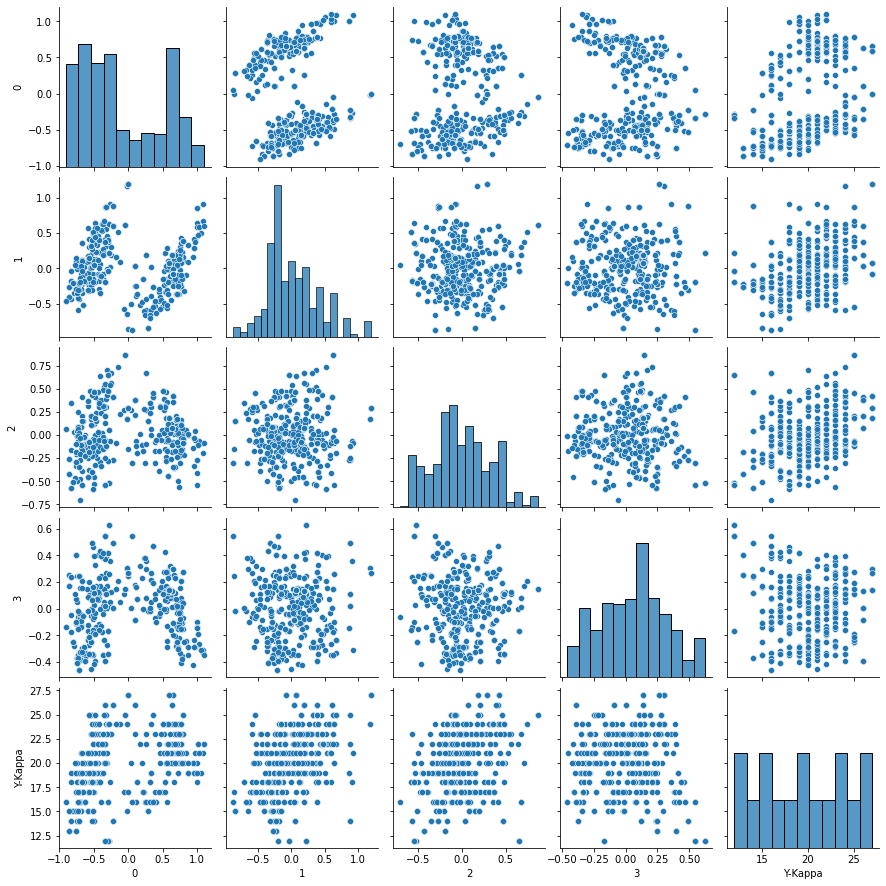


**Accuracy analysis(RandomForest model):**

****

****

**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%**

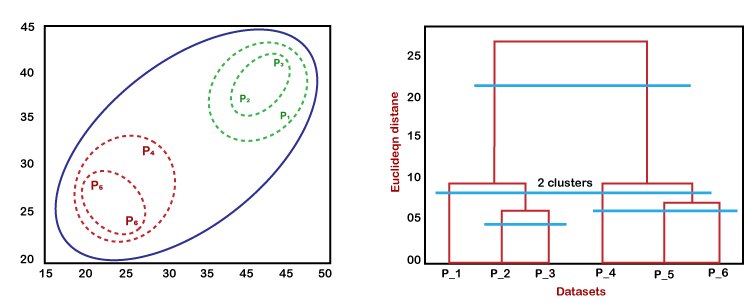
**Q2: (Hierarchical Clustering)**

**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')

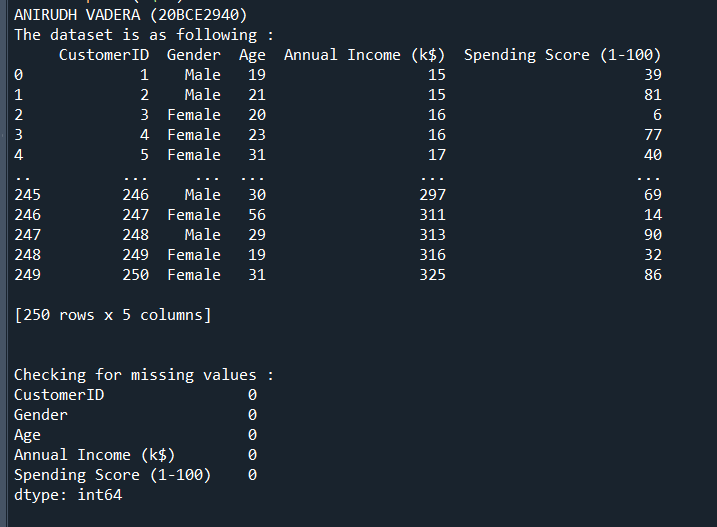
mtp.show()

print("Sum of squared error: %.2f" % (sum(pow(y\_pred-y\_test,2))))

**Output and Results:**

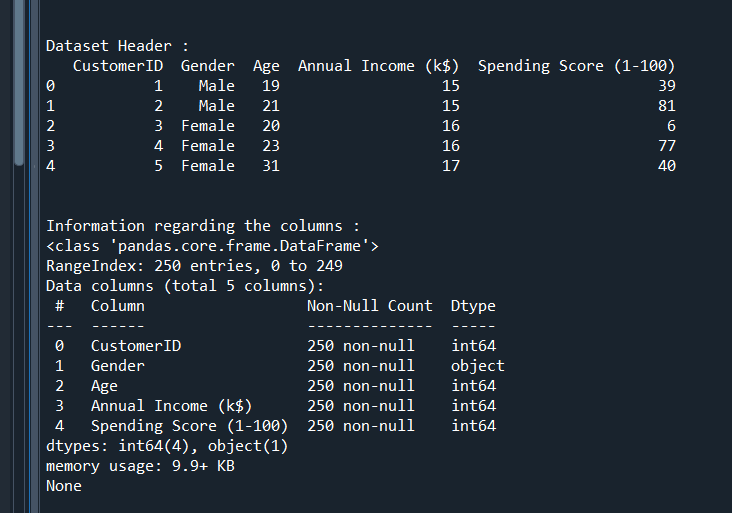
**Data Pre-processing:**

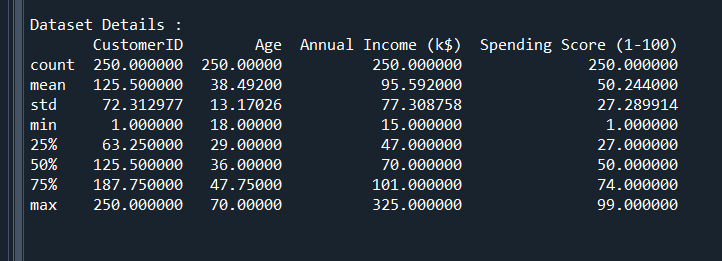
**Dataset:**

****

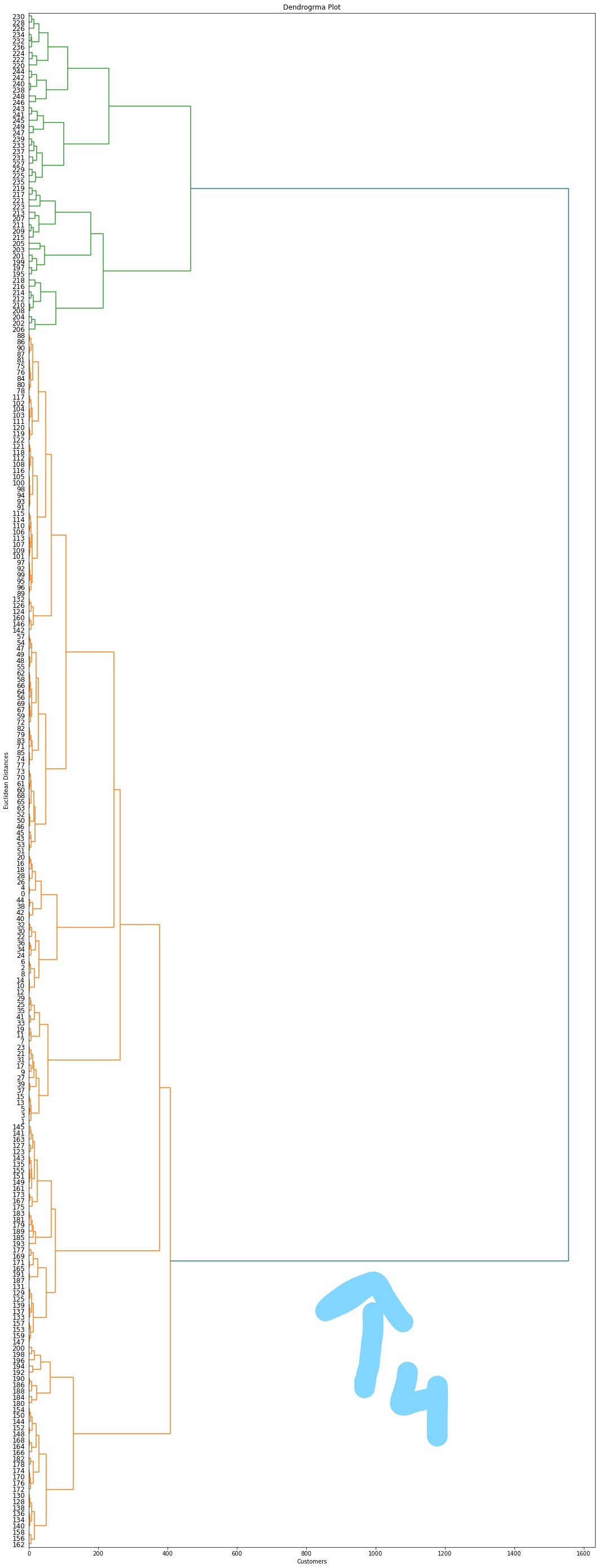
**There are no missing values therefore we can move forward:**

**Dataset Details:**





**Finding the optimal number of clusters using the Dendrogram:**

****

**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)

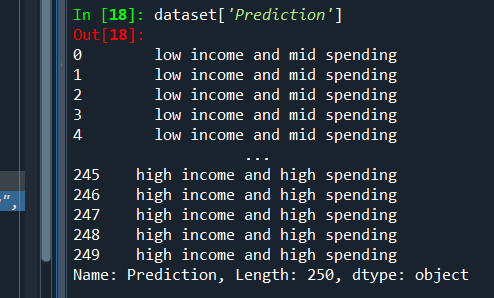
**So, the optimal number of clusters will be 5**, and we will train the model in the next step, using the same.

**Training the hierarchical clustering model:**

**Parameters:**

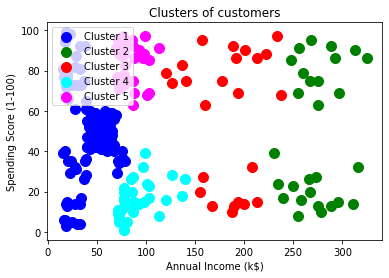
* **n\_clusters=5**
* **affinity='euclidean'**: It is a metric used to compute the linkage.
* **linkage='ward'**

**The Y\_test is as follows:**

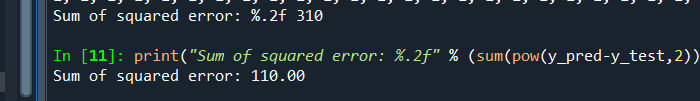


**Visualizing the clusters:**

* 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