# 

Addis Ababa University

College of Natural and Computational Sciences

School Of Information Science

MSc. In Information Science and Systems (Information Science)

***An Implementation of Agglomerative Clustering:***

Module: **Applied Machine Learning**

Submitted by: Selahadin Nurga

Id No: GSE/7245/13

Submitted to: Dr. Michael (PHD)

February: 2022

**Implementation of Agglomerative Clustering with Scikit-Learn**

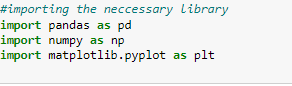
[Agglomerative clustering](https://www.codespeedy.com/introduction-to-hierarchical-clustering-algorithm/)is a technique in which we cluster the data into classes in a hierarchical manner. In this technique we can start using a top-down approach or a bottom-up approach. In the case of bottom-up approach, all data points are treated as unique clusters at the start. Then, in each iteration, the algorithm merges the two closest clusters into a single cluster. This process continues until we reach the required number of clusters.

This article tries to shows the implementation of hierarchical clustering analysis using Python and the scikit-learn library.

A scikit-learn provides an Agglomerative Clustering class or packages that used to implement the agglomerative clustering algorithm. It has several parameters to set. The linkage parameter defines the merging criteria that the distance method between the sets of the observation data.  The "ward", "complete", "average", and "single" methods can be used. Affinity parameter defines the distance metric to compute the linkage. The number of clusters can be set with the ‘n\_clusters’ parameter.

To start off, the necessary libraries are imported. Those initially imported libraries are used us for scientific computing, plotting and many other data preprocessing

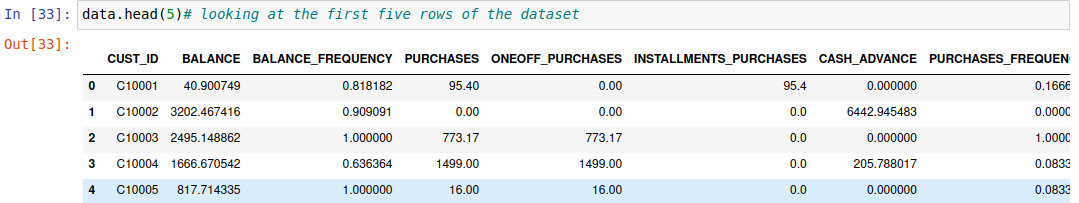
**Importing required libraries**



After importing the libraries, we follow this by loading the dataset and split into input and output.

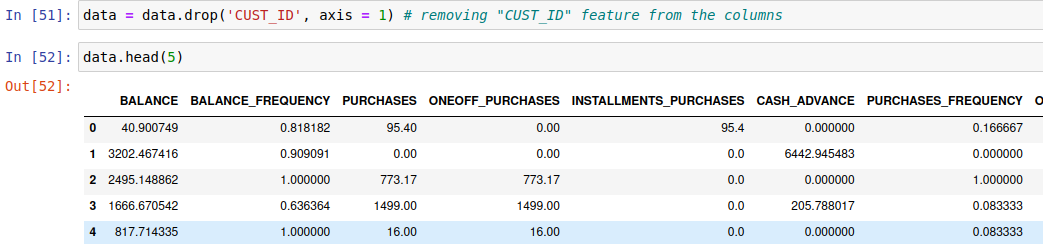
**Loading and cleaning the data**





Now drop the feature which have no impact for the model. In this data set drop the “CUST\_ID” from the data using drop() function.

data **=** data.drop('CUST\_ID', axis **=** 1)



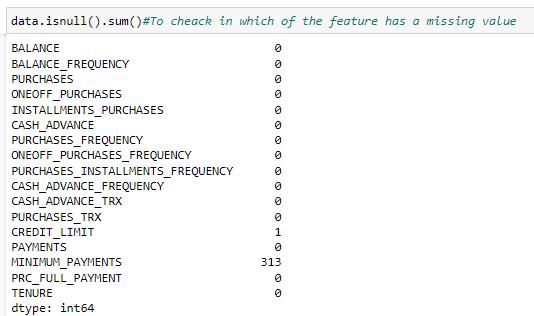
**Handling the missing values**

First we have to check whether there are a missing values or not in the data set.

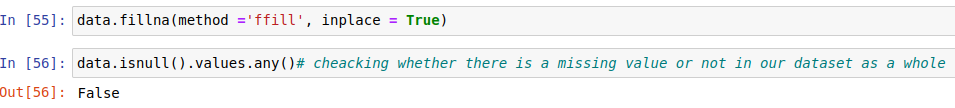
data.isnull().values.any()

As we can see from the above result, we have a missing value in our dataset and we have to check in which of our features have that missing value using the below code.

data.isnull().sum()



From the above code result, the data set with a feature name “MINIMUM\_PAYMENTS” and “CREDIT\_LIMIT” have a missing value, we have to handle those missing values using different methods. For this model ‘ffill’ method is used. ffill or forward-fill propagates the last observed non-null value forward until another non-null value is encountered.

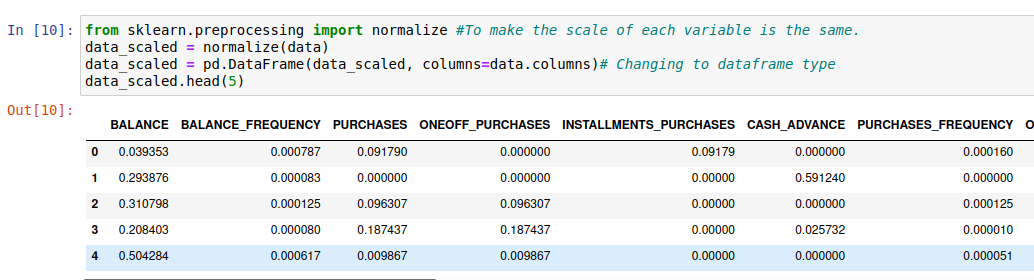
**data.fillna(method ='ffill', inplace = True)**

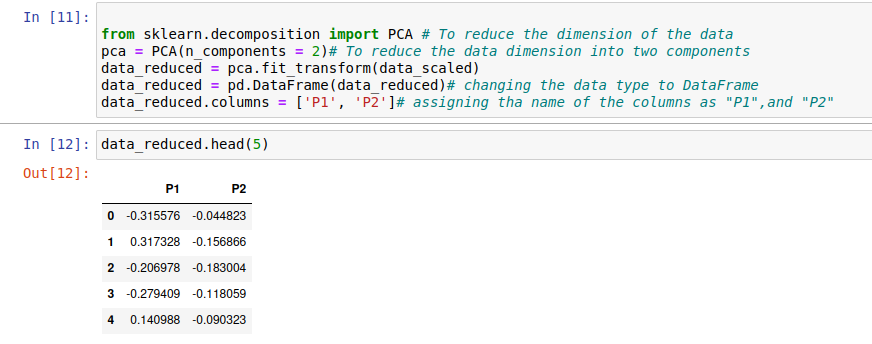
But before applying Hierarchical Clustering, we have to normalize the data so that the scale of each variable is the same. This is important because if the scale of the variables is not the same, the model might become biased towards the variables with a higher magnitude

So, let’s first normalize the data and bring all the variables to the same scale:

|  |
| --- |
| from sklearn.preprocessing import normalize |
| data\_scaled = normalize(data) |
| data\_scaled = pd.DataFrame(data\_scaled, columns=data.columns) |

data\_scaled.head(5)

Here, we can see that the scale of all the variables is almost similar. Now, we are good to go.

Now let us reduce the dimension of the datasets as It is used when we need to tackle the curse of dimensionality among data with linear relationships, i.e. where having too many dimensions (features) in our data causes noise and difficulties (it can be sound, picture, or context). 

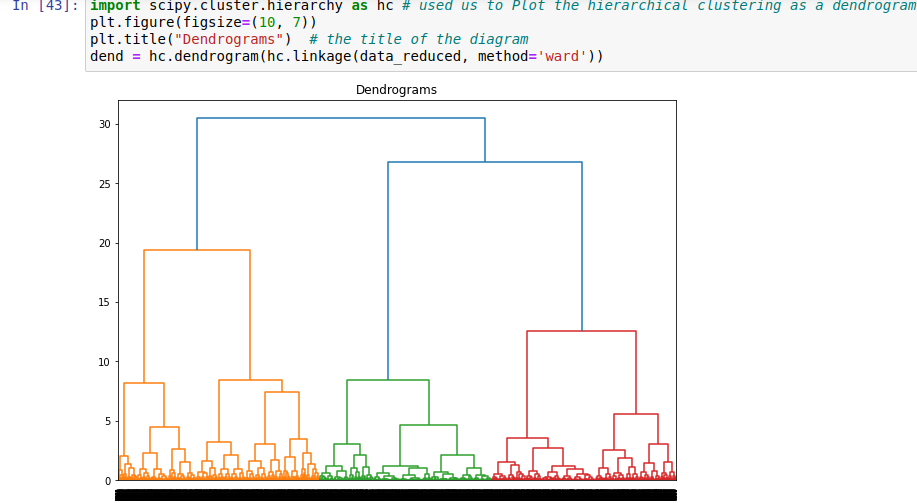
Let’s first draw the dendrogram to help us decide the number of clusters for this particular problem by importing a necessary libraries for dendograms. Dendograms are used to divide a given cluster into many different clusters.

import scipy.cluster.hierarchy as hc

plt.figure(figsize=(10, 7))

plt.title("Dendrograms")

dend = hc.dendrogram(hc.linkage(data\_reduced, method='ward'))



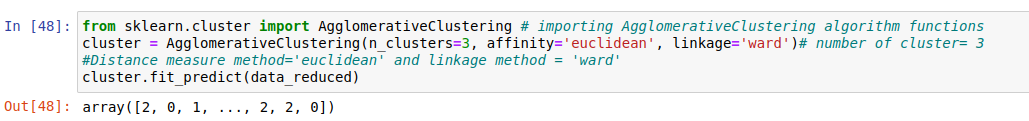
The x-axis contains the samples and y-axis represents the distance between these samples. The vertical line with maximum distance is the blue line and hence we can decide a threshold of 6 and cut the dendrogram:

We have three clusters as this line cuts the dendrogram at three points. Let’s now apply hierarchical clustering for 3 clusters:

from sklearn.cluster import AgglomerativeClustering

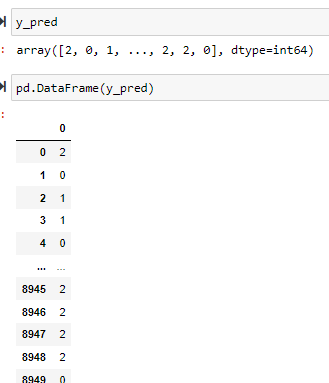
cluster = AgglomerativeClustering(n\_clusters=3, affinity='euclidean', linkage='ward')

cluster.fit\_predict(data\_reduced)

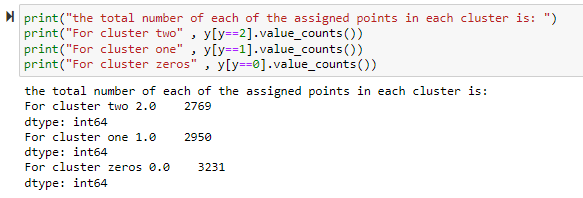


We can see from the output of the above code, the values of 0s ,1s and 2s in the output since we defined 3 clusters. 0 represents the points that belong to the first cluster and 1 represents points in the second cluster and 3 represents points in the third cluster.

To look which of the instance in each rows are assigned to which cluster, we can do like the below code by converting the array data type in to pandas Data Frame. As we can see from the result the first row is assigned to cluster 2, second row to cluster 0 and third row to cluster 1 and so on.



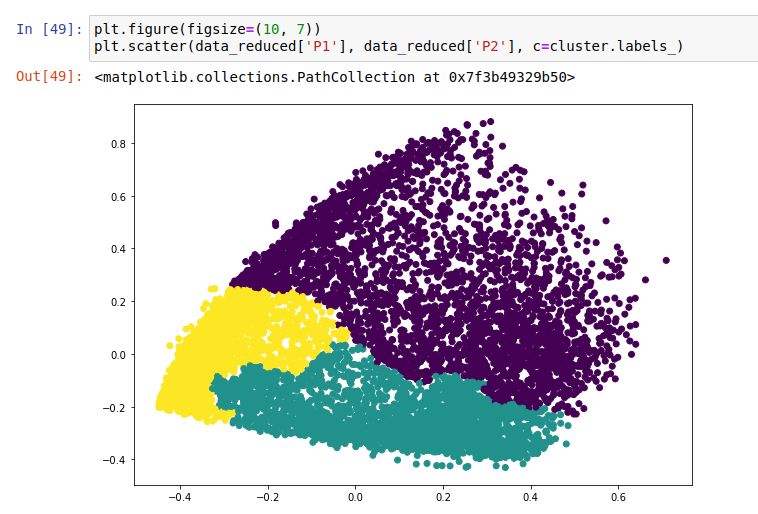
We can also know how many of the instance are assigned in each of the three cluster using the code below.



So cluster 0 have a total of 3231 instance, cluster 1 2950 instance and cluster 2 2769 instance. Let’s now visualize the two clusters:

plt.figure(figsize=(10, 7))

plt.scatter(data\_reduced['P1'], data\_reduced['P2'], c=cluster.labels\_)



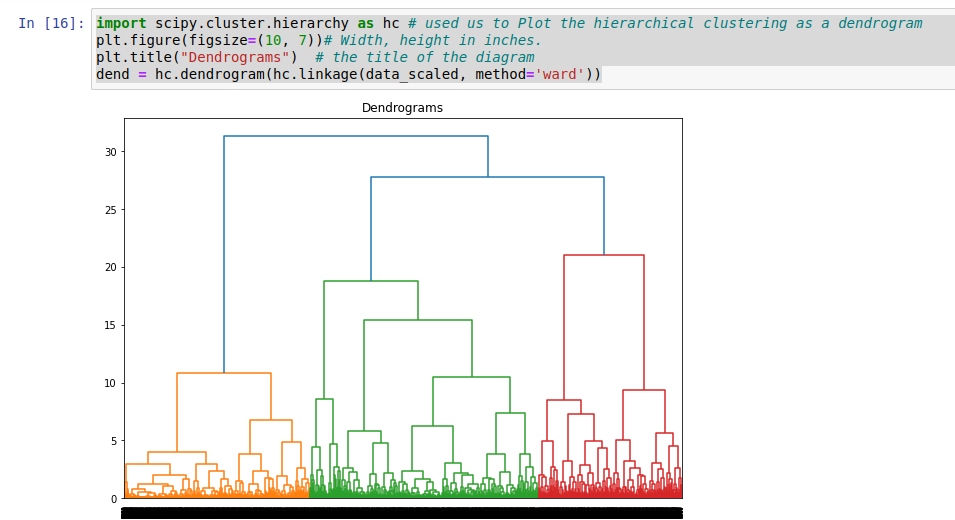
# With out Applying PCA

import scipy.cluster.hierarchy as hc

plt.figure(figsize=(10, 7))

plt.title("Dendrograms")

dend = hc.dendrogram(hc.linkage(data\_scaled, method='ward'))

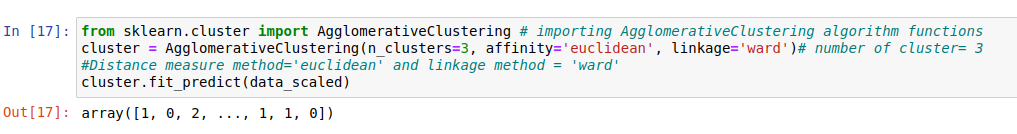


Let’s now apply hierarchical clustering for 3 clusters:

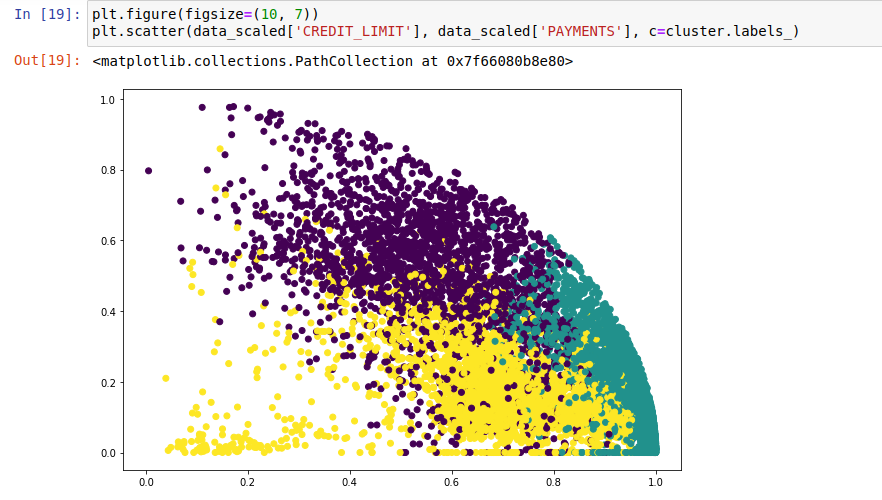
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n\_clusters=3, affinity='euclidean', linkage='ward')

cluster.fit\_predict(data\_scaled)



Let’s now visualize the two clusters of 0s, 1s, and 2s in scatter ploting



**Conclusion**

Agglomerative clustering is a super useful way of segmenting observation. This algorithm has Strength as it does not have to assume any particular number of clusters. Any desired number of clusters can be obtained by ‘cutting’ the dendogram at the proper level. One thing that we take from this algorithm is it do not scale well and the time complexity of at least O(n2), where the total number of data objects is n.