Unit 7 : Clustering

K-Means Clustering Algorithm: Applications, Types, and How Does It Work?

Clustering is a type of unsupervised learning wherein data points are grouped into different sets based on their degree of similarity.
The various types of clustering are:

- Hierarchical clustering
- Partitioning clustering

Hierarchical clustering is further subdivided into:

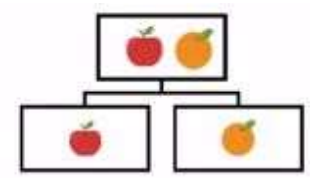
- Agglomerative clustering
- Divisive clustering

Partitioning clustering is further subdivided into:

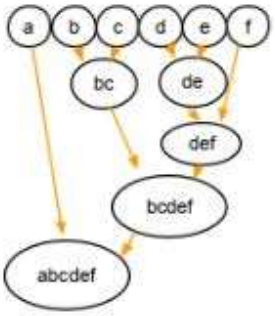
- K-Means clustering
- Fuzzy C-Means clustering

Hierarchical Clustering

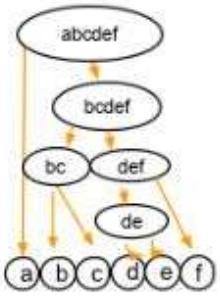
Hierarchical clustering uses a tree-like structure, like so:



In agglomerative clustering, there is a bottom-up approach. We begin with each element as a separate cluster and merge them into successively more massive clusters, as shown below:



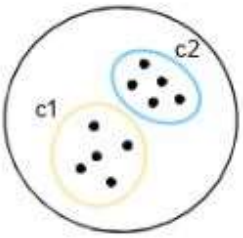
Divisive clustering is a top-down approach. We begin with the whole set and proceed to divide it into successively smaller clusters, as you can see below:



Partitioning Clustering

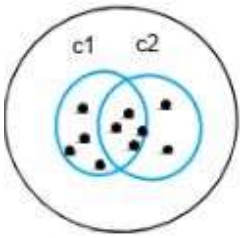
Partitioning clustering is split into two subtypes - K-Means clustering and Fuzzy C-Means.

In k-means clustering, the objects are divided into several clusters mentioned by the number 'K.' So if we say K = 2, the objects are divided into two clusters, c1 and c2, as shown:



Here, the features or characteristics are compared, and all objects having similar characteristics are clustered together.

Fuzzy c-means is very similar to k-means in the sense that it clusters objects that have similar characteristics together. In k-means clustering, a single object cannot belong to two different clusters. But in c-means, objects can belong to more than one cluster, as shown.



What is Meant by the K-Means Clustering Algorithm?

K-Means clustering is an unsupervised learning algorithm. There is no labeled data for this clustering, unlike in supervised learning. K-Means performs the division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

The term ‘K’ is a number. You need to tell the system how many clusters you need to create. For example, K = 2 refers to two clusters. There is a way of finding out what is the best or optimum value of K for a given data.

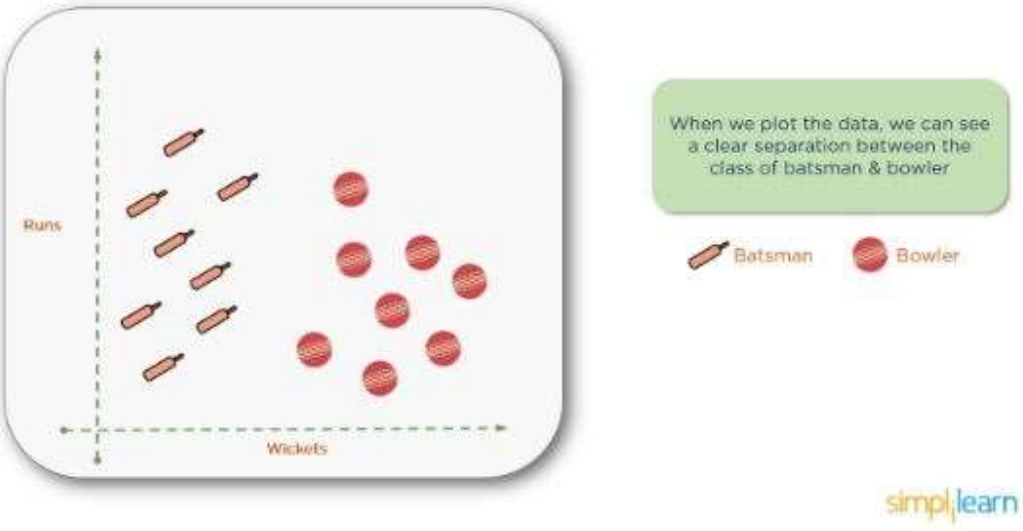
For a better understanding of k-means, let's take an example from cricket. Imagine you received data on a lot of cricket players from all over the world, which gives information on the runs scored by the player and the wickets taken by them in the last ten matches. Based on this information, we need to group the data into two clusters, namely batsman and bowlers.

Solution:

Assign data points

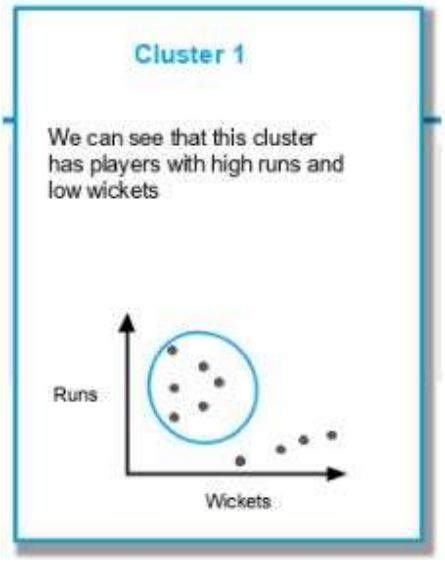
Here, we have our data set plotted on ‘x’ and ‘y’ coordinates. The information on the y-axis is about the runs scored, and on the x-axis about the wickets taken by the players.

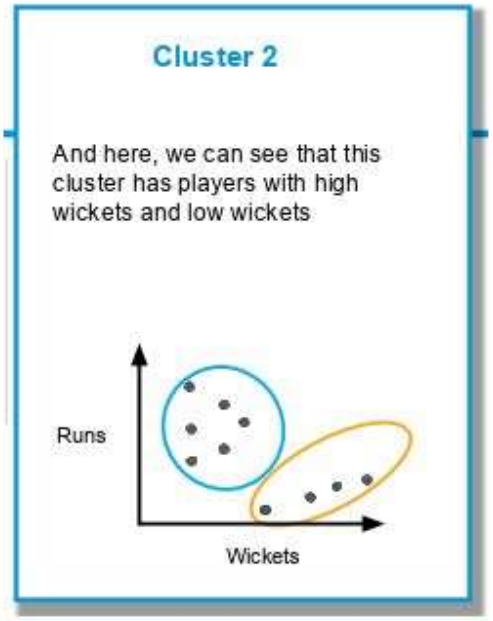
If we plot the data, this is how it would look:



Perform Clustering

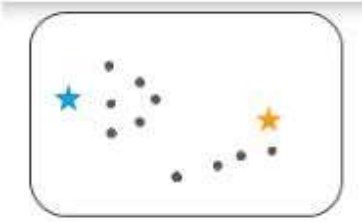
We need to create the clusters, as shown below:





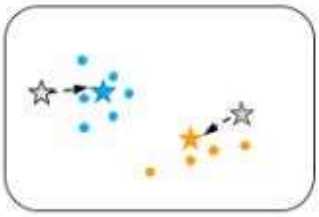
Considering the same data set, let us solve the problem using K-Means clustering (taking K = 2).

The first step in k-means clustering is the allocation of two centroids randomly (as K=2). Two points are assigned as centroids. Note that the points can be anywhere, as they are random points. They are called centroids, but initially, they are not the central point of a given data set.



The next step is to determine the distance between each of the randomly assigned centroids' data points. For every point, the distance is measured from both the centroids, and whichever distance is less, that point is assigned to that centroid. You can see the data points attached to the centroids and represented here in blue and yellow.

The next step is to determine the actual centroid for these two clusters. The original randomly allocated centroid is to be repositioned to the actual centroid of the clusters.



This process of calculating the distance and repositioning the centroid continues until we obtain our final cluster. Then the centroid repositioning stops.

As seen above, the centroid doesn't need anymore repositioning, and it means the algorithm has converged, and we have the two clusters with a centroid.

Applications of K-Means Clustering

K-Means clustering is used in a variety of examples or business cases in real life, like:

- Academic performance
- Diagnostic systems
- Search engines
- Wireless sensor networks

Academic Performance

Based on the scores, students are categorized into grades like A, B, or C.

Diagnostic systems

The medical profession uses k-means in creating smarter medical decision support systems, especially in the treatment of liver ailments.

Search engines

Clustering forms a backbone of search engines. When a search is performed, the search results need to be grouped, and the search engines very often use clustering to do this.

Wireless sensor networks

The clustering algorithm plays the role of finding the cluster heads, which collect all the data in its respective cluster.

Distance Measure

Distance measure determines the similarity between two elements and influences the shape of clusters.

K-Means clustering supports various kinds of distance measures, such as:

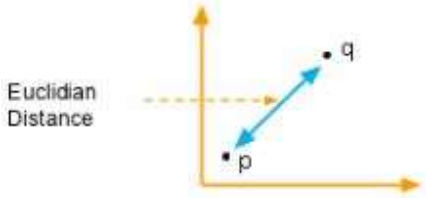
- Euclidean distance measure
- Manhattan distance measure
- A squared euclidean distance measure
- Cosine distance measure

Euclidean Distance Measure

The most common case is determining the distance between two points. If we have a point P and point Q, the euclidean distance is an ordinary straight line. It is the distance between the two points in Euclidean space.

The formula for distance between two points is shown below:

$$d=\sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$



Squared Euclidean Distance Measure

This is identical to the Euclidean distance measurement but does not take the square root at the end. The formula is shown below:

$$d=\sum_{i=1}^{n}(q_i-p_i)^2$$

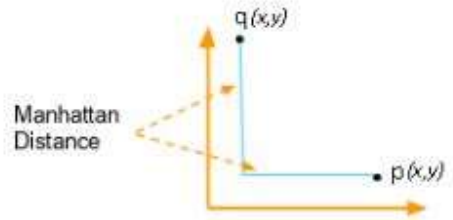
Manhattan Distance Measure

The Manhattan distance is the simple sum of the horizontal and vertical components or the distance between two points measured along axes at right angles.

Note that we are taking the absolute value so that the negative values don't come into play.

The formula is shown below:

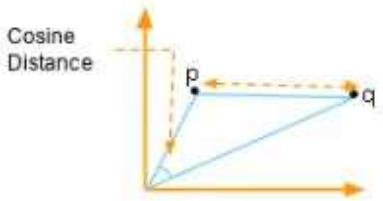
$$d = \sum_{i=1}^{n} |q_x - p_x| + |q_x - p_y|$$



Cosine Distance Measure

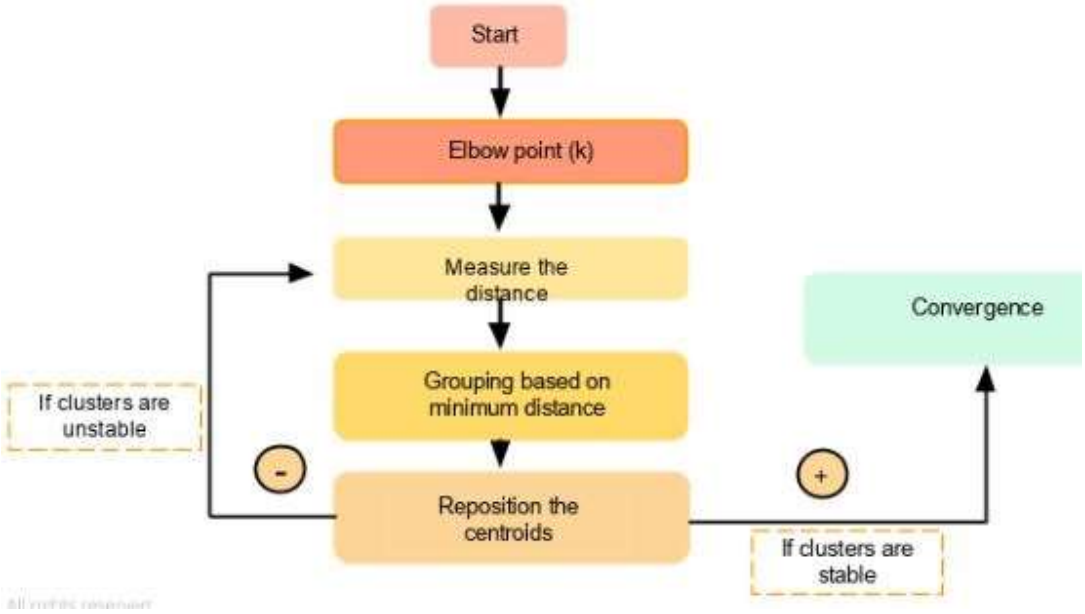
In this case, we take the angle between the two vectors formed by joining the origin point. The formula is shown below:

$$d = \frac{\sum_{i=0}^{n-1} q_i - p_x}{\sum_{i=0}^{n-1} (q_i)^2 \times \sum_{i=0}^{n-1} (p_i)^2}$$



How Does K-Means Clustering Work?

The flowchart below shows how k-means clustering works:



The goal of the K-Means algorithm is to find clusters in the given input data. There are a couple of ways to accomplish this. We can use the trial and error method by specifying the value of K (e.g., 3,4, 5). As we progress, we keep changing the value until we get the best clusters.

Another method is to use the Elbow technique to determine the value of K. Once we get the K's value, the system will assign that many centroids randomly and measure the distance of each of the data points from these centroids. Accordingly, it assigns those points to the corresponding centroid from which the distance is minimum. So each data point will be assigned to the centroid, which is closest to it. Thereby we have a K number of initial clusters.

For the newly formed clusters, it calculates the new centroid position. The position of the centroid moves compared to the randomly allocated one.

Once again, the distance of each point is measured from this new centroid point. If required, the data points are relocated to the new centroids, and the mean position or the new centroid is calculated once again.

If the centroid moves, the iteration continues indicating no convergence. But once the centroid stops moving (which means that the clustering process has converged), it will reflect the result.

Let's use a visualization example to understand this better.

We have a data set for a grocery shop, and we want to find out how many clusters this has to be spread across. To find the optimum number of clusters, we break it down into the following steps:

Step 1:

The Elbow method is the best way to find the number of clusters. The elbow method constitutes running K-Means clustering on the dataset.

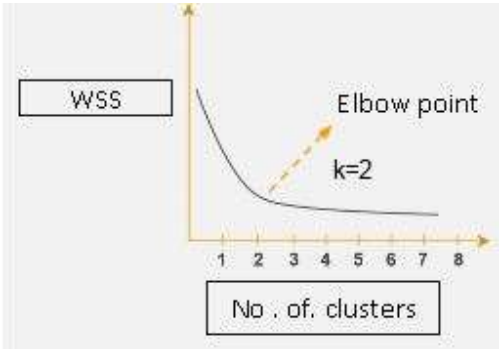
Next, we use within-sum-of-squares as a measure to find the optimum number of clusters that can be formed for a given data set. Within the sum of squares (WSS) is defined as the sum of the squared distance between each member of the cluster and its centroid.

$$WSS = \sum_{i=1}^{m} (x_i - c_i)^2$$

Where x_i = data point and c_i = closest point to centroid

The WSS is measured for each value of K. The value of K, which has the least amount of WSS, is taken as the optimum value.

Now, we draw a curve between WSS and the number of clusters.



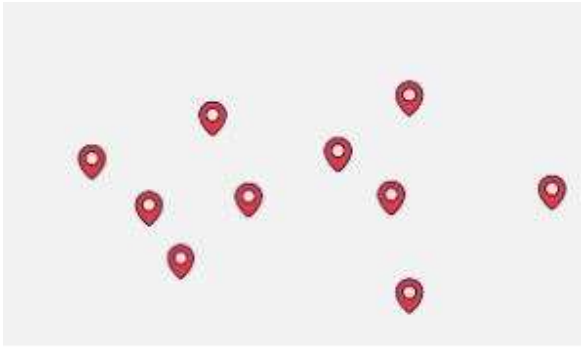
Here, WSS is on the y-axis and number of clusters on the x-axis.

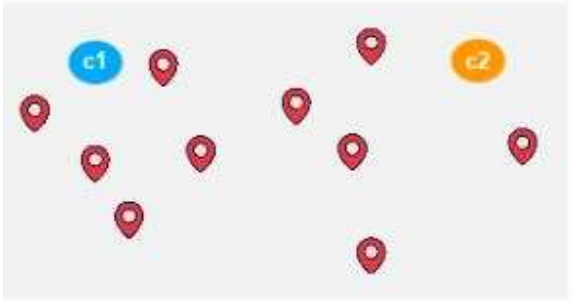
You can see that there is a very gradual change in the value of WSS as the K value increases from 2.

So, you can take the elbow point value as the optimal value of K. It should be either two, three, or at most four. But, beyond that, increasing the number of clusters does not dramatically change the value in WSS, it gets stabilized.

Step 2:

Let's assume that these are our delivery points:





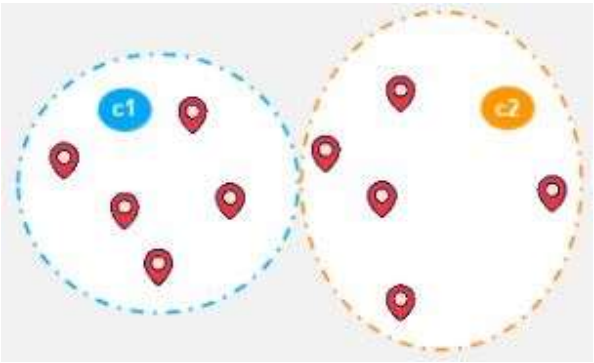
We can randomly initialize two points called the cluster centroids.

Here, C1 and C2 are the centroids assigned randomly.

Step 3:

Now the distance of each location from the centroid is measured, and each data point is assigned to the centroid, which is closest to it.

This is how the initial grouping is done:



Step 4:

Compute the actual centroid of data points for the first group.

Step 5:

Reposition the random centroid to the actual centroid.

Step 6:

Compute the actual centroid of data points for the second group.

Step 7:

Reposition the random centroid to the actual centroid.

Step 8:

Once the cluster becomes static, the k-means algorithm is said to be converged.

The final cluster with centroids c1 and c2 is as shown below:

K-Means Clustering Algorithm

Let's say we have x1, x2, x3..... x(n) as our inputs, and we want to split this into K clusters.

The steps to form clusters are:

Step 1: Choose K random points as cluster centers called centroids.

Step 2: Assign each x(i) to the closest cluster by implementing euclidean distance (i.e., calculating its distance to each centroid)

Step 3: Identify new centroids by taking the average of the assigned points.

Step 4: Keep repeating step 2 and step 3 until convergence is achieved

Let's take a detailed look at it at each of these steps.

Step 1:

We randomly pick K (centroids). We name them c1,c2,..... ck, and we can say that

$$C = c_1, c_2, \ldots, c_k$$

Where C is the set of all centroids.

Step 2:

We assign each data point to its nearest center, which is accomplished by calculating the euclidean distance.

$$\arg\min_{c_i \in C} dist(c_i, x)^2$$

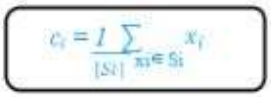
Where dist() is the Euclidean distance.

Here, we calculate each x value's distance from each c value, i.e. the distance between x1-c1, x1-c2, x1-c3, and so on. Then we find which is the lowest value and assign x1 to that particular centroid.

Similarly, we find the minimum distance for x2, x3, etc.

Step 3:

We identify the actual centroid by taking the average of all the points assigned to that cluster.



Where Si is the set of all points assigned to the ith cluster.

It means the original point, which we thought was the centroid, will shift to the new position, which is the actual centroid for each of these groups.

Step 4:

Keep repeating step 2 and step 3 until convergence is achieved.

How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{P_i \text{ in Cluster1}} distance(P_i\ C_1)^2 + \sum_{P_i \text{ in Cluster2}} distance(P_i\ C_2)^2 + \sum_{P_i \text{ in CLuster3}} distance(P_i\ C_3)^2$$

In the above formula of WCSS,

$\sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculate the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Hierarchical Clustering in Machine Learning

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**.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

1. **Agglomerative:** Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

Why hierarchical clustering?

As we already have other clustering algorithms such as **K-Means Clustering**, then why we need hierarchical clustering? So, as we have seen in the K-means clustering that there are some challenges with this algorithm, which are a predetermined number of clusters, and it always tries to create the clusters of the same size. To solve these two challenges, we can opt for the hierarchical clustering algorithm because, in this algorithm, we don't need to have knowledge about the predefined number of clusters.

In this topic, we will discuss the Agglomerative Hierarchical clustering algorithm.

Agglomerative Hierarchical clustering

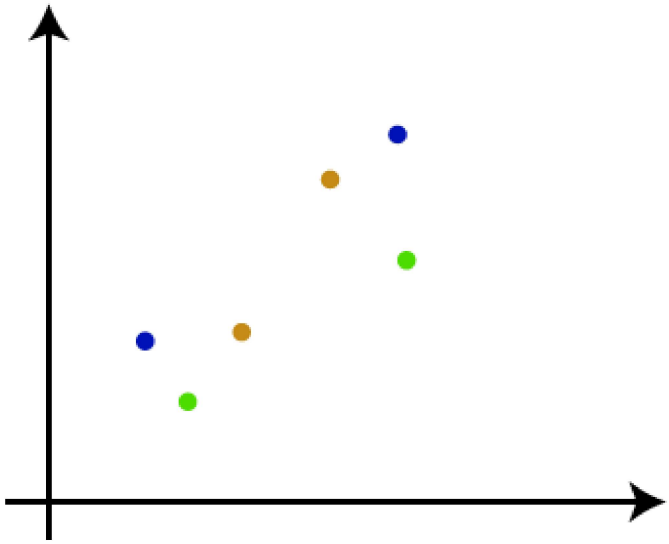
The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the **bottom-up approach**. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.

This hierarchy of clusters is represented in the form of the dendrogram.

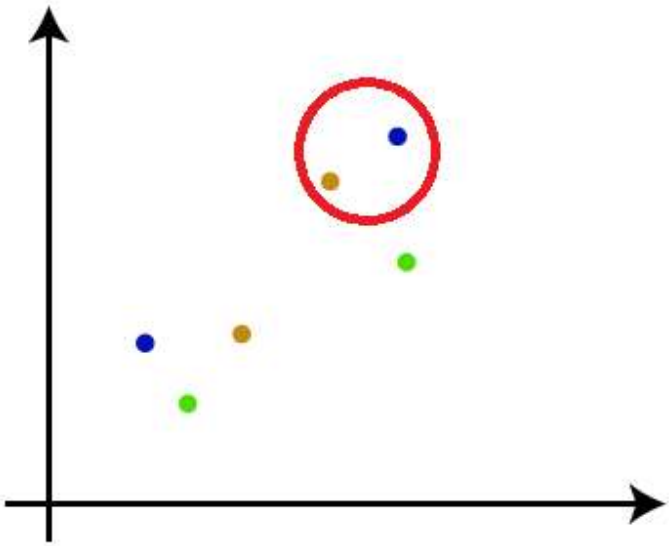
How the Agglomerative Hierarchical clustering Work?

The working of the AHC algorithm can be explained using the below steps:

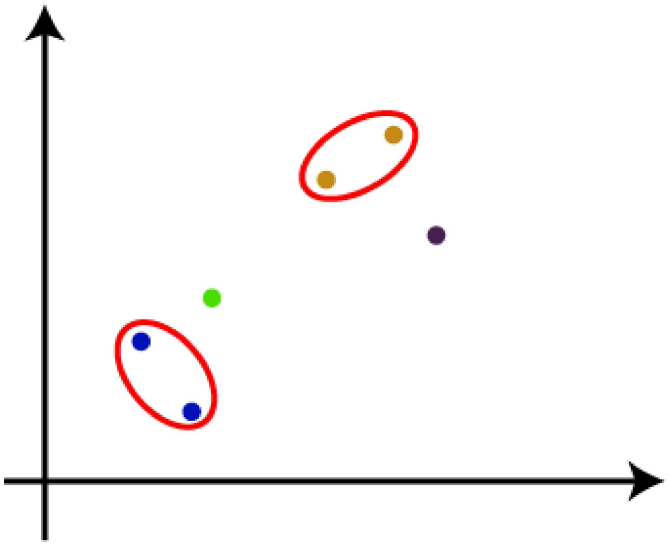
- **Step-1:** Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.



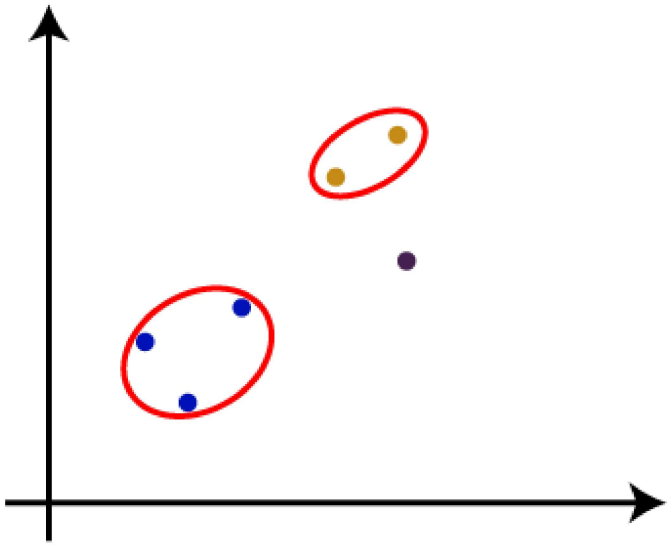
Step-2: Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.

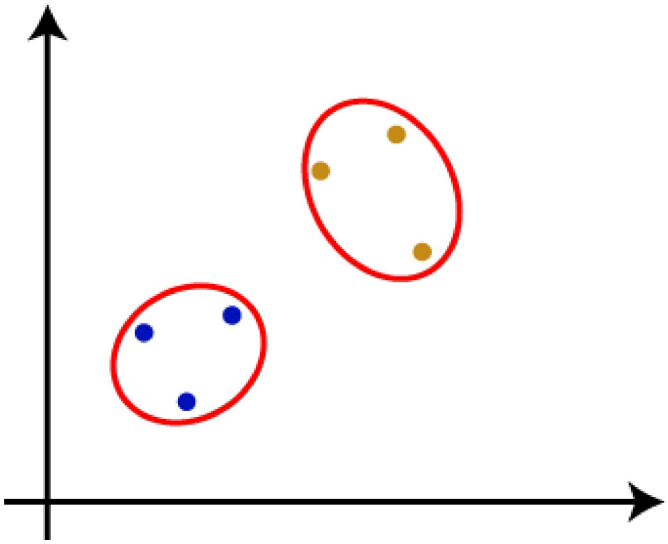


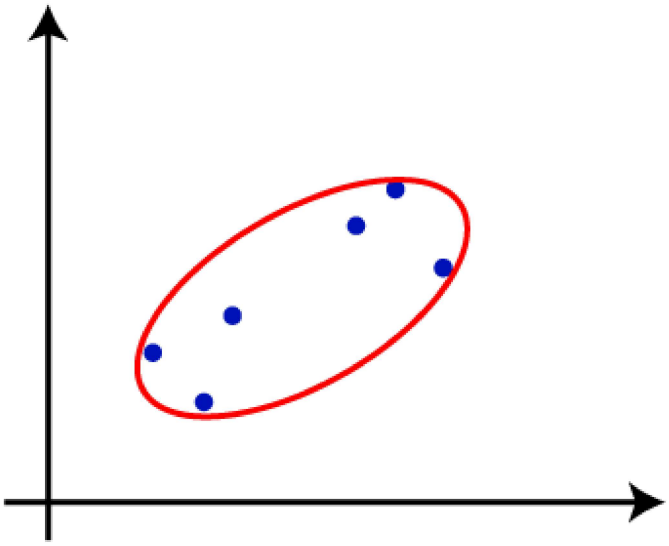
Step-3: Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.



Step-4: Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:





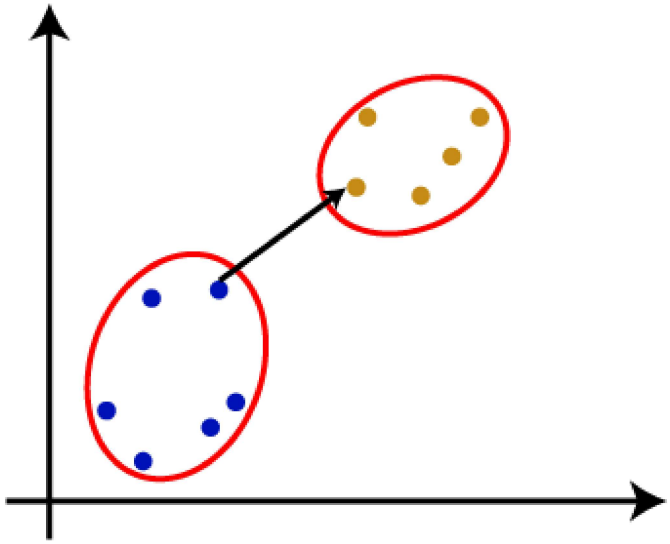


- **Step-5:** Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

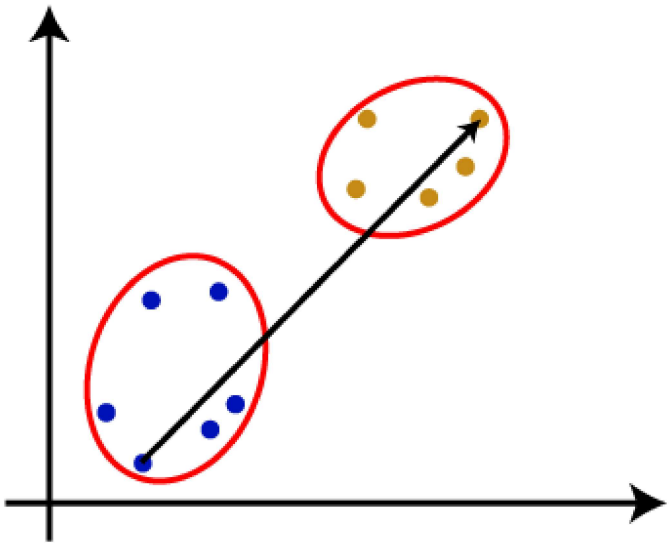
Measure for the distance between two clusters

As we have seen, the **closest distance** between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering. These measures are called **Linkage methods**. Some of the popular linkage methods are given below:

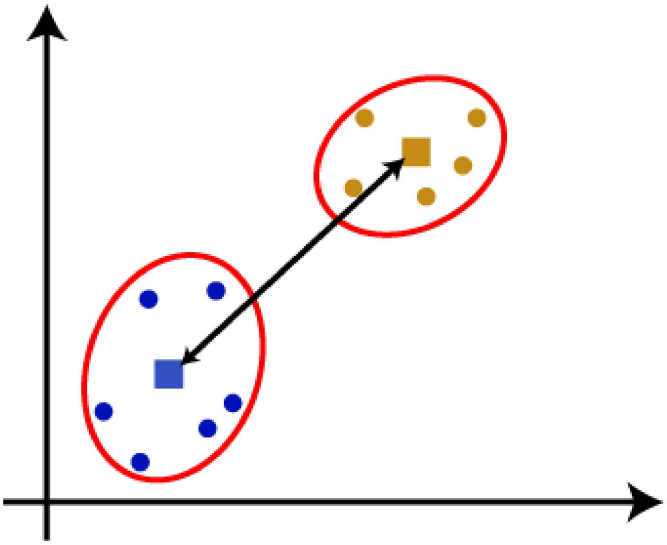
1. **Single Linkage:** It is the Shortest Distance between the closest points of the clusters. Consider the below image:



Complete Linkage: It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.



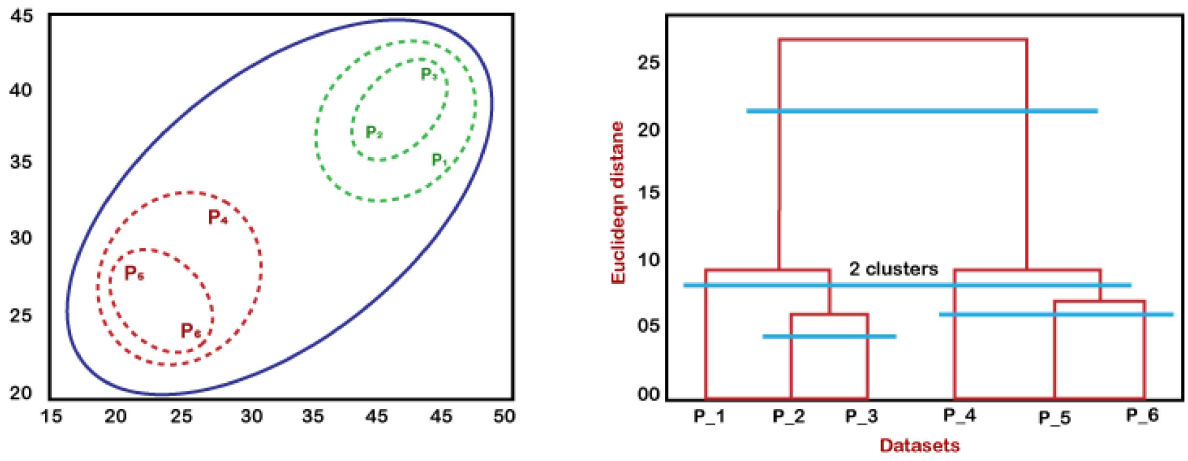
1. **Average Linkage:** It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.
2. **Centroid Linkage:** It is the linkage method in which the distance between the centroid of the clusters is calculated. Consider the below image:



Woking of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

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.

- As we have discussed above, 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.

Python Implementation of Agglomerative Hierarchical Clustering

Now we will see the practical implementation of the agglomerative hierarchical clustering algorithm using Python. To implement this, we will use the same dataset problem that we have used in the previous topic of K-means clustering so that we can compare both concepts easily.

The dataset contains the information of customers that have visited a mall for shopping. So, the mall owner wants to find some patterns or some particular behavior of his customers using the dataset information.

Steps for implementation of AHC using Python:

The steps for implementation will be the same as the k-means clustering, except for some changes such as the method to find the number of clusters. Below are the steps:

1. **Data Pre-processing**
2. **Finding the optimal number of clusters using the Dendrogram**
3. **Training the hierarchical clustering model**
4. **Visualizing the clusters**

Data Pre-processing Steps:

In this step, we will import the libraries and datasets for our model.

- **Importing the libraries**

```
# Importing the libraries
import numpy as nm
```

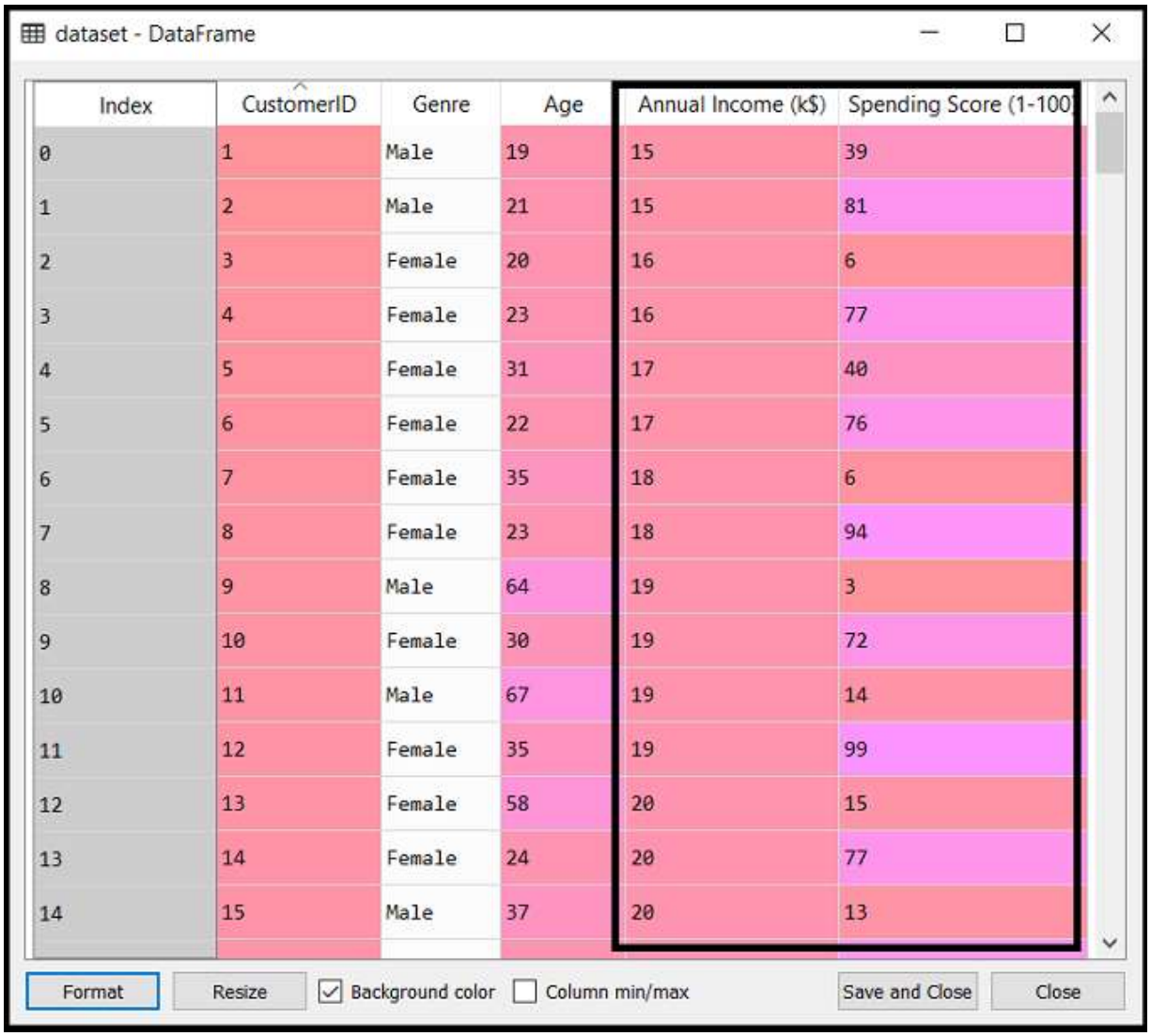
```
3. import matplotlib.pyplot as mtp
4. import pandas as pd
```

The above lines of code are used to import the libraries to perform specific tasks, such as **numpy** for the Mathematical operations, **matplotlib** for drawing the graphs or scatter plot, and **pandas** for importing the dataset.

- **Importing the dataset**

```
1. # Importing the dataset
2. dataset = pd.read_csv('Mall_Customers_data.csv')
```

As discussed above, we have imported the same dataset of **Mall_Customers_data.csv,** as we did in k-means clustering. Consider the below output:



Index	CustomerID	Genre	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
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13

- **Extracting the matrix of features**

Here we will extract only the matrix of features as we don't have any further information about the dependent variable. Code is given below:

```
x = dataset.iloc[:, [3, 4]].values
```

Here we have extracted only 3 and 4 columns as we will use a 2D plot to see the clusters. So, we are considering the Annual income and spending score as the matrix of features.

Step-2: Finding the optimal number of clusters using the Dendrogram

Now we will find the optimal number of clusters using the Dendrogram for our model. For this, we are going to use **scipy** library as it provides a function that will directly return the dendrogram for our code. Consider the below lines of code:

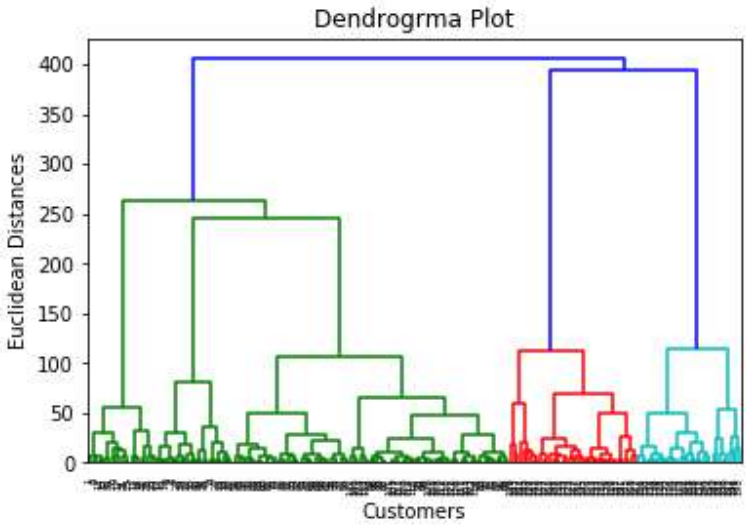
```
1. #Finding the optimal number of clusters using the dendrogram
2. import scipy.cluster.hierarchy as shc
3. dendro = shc.dendrogram(shc.linkage(x, method="ward"))
4. mtp.title("Dendrogrma Plot")
5. mtp.ylabel("Euclidean Distances")
6. mtp.xlabel("Customers")
7. mtp.show()
```

In the above lines of code, we have imported the **hierarchy** module of scipy library. This module provides us a method **shc.denrogram(),** which takes the **linkage()** as a parameter. The linkage function is used to define the distance between two clusters, so here we have passed the x(matrix of features), and method "**ward**," the popular method of linkage in hierarchical clustering.

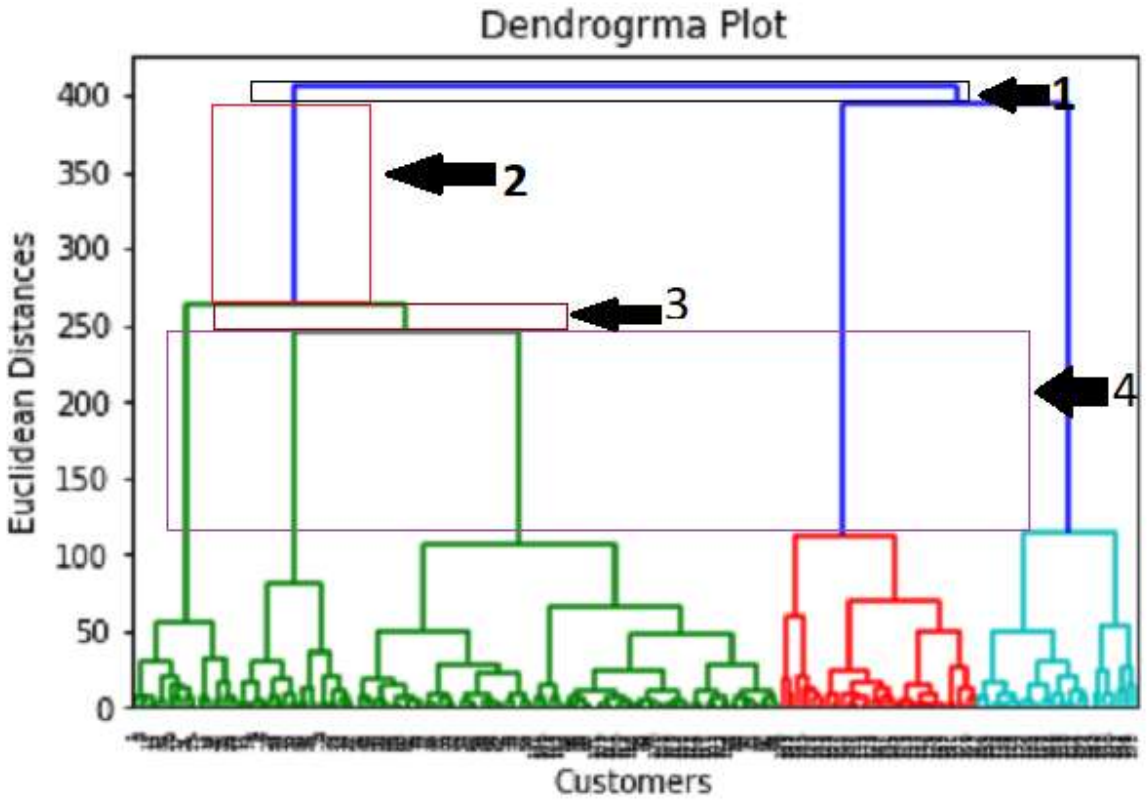
The remaining lines of code are to describe the labels for the dendrogram plot.

Output:

By executing the above lines of code, we will get the below output**:**



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. Consider the below diagram:



In the above diagram, we have shown the vertical distances that are not cutting their horizontal bars. 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). We can also take the 2nd number as it approximately equals the 4th distance, but we will consider the 5 clusters because the same we calculated in the K-means algorithm.

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

Step-3: Training the hierarchical clustering model

As we know the required optimal number of clusters, we can now train our model. The code is given below:

```
1. #training the hierarchical model on dataset
2. from sklearn.cluster import AgglomerativeClustering
3. hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
4. y_pred= hc.fit_predict(x)
```

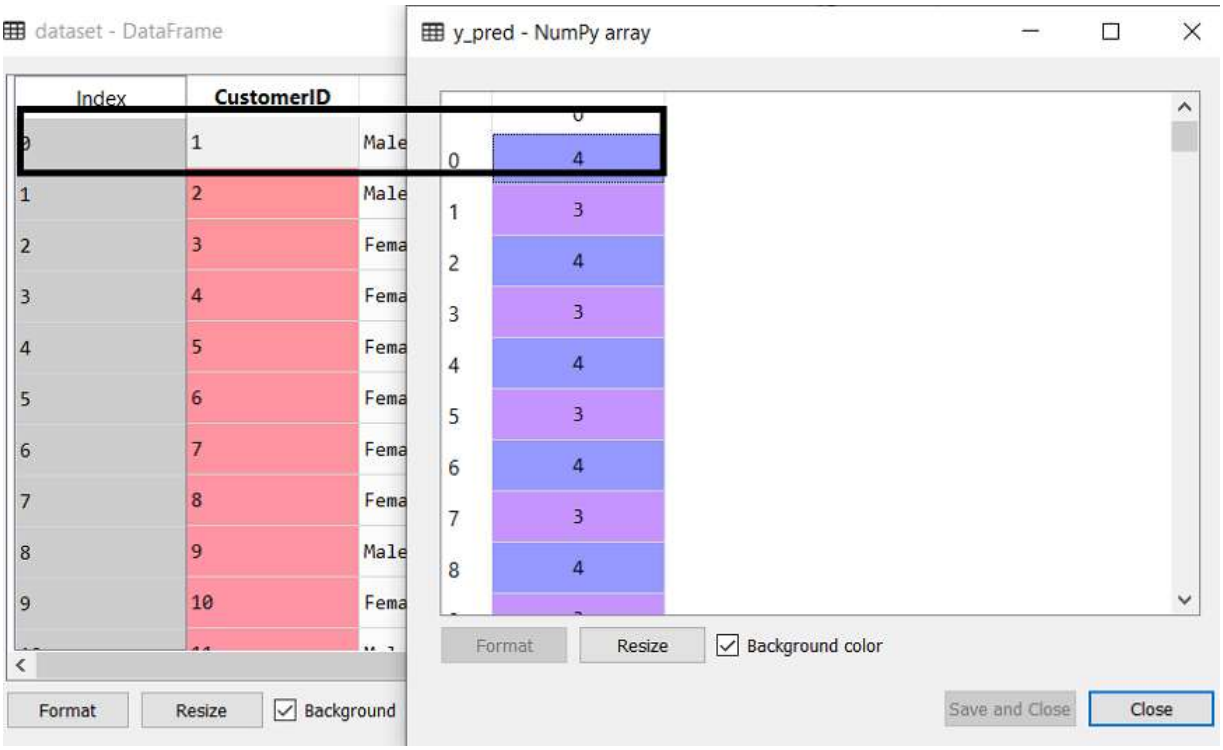
In the above code, we have imported the **AgglomerativeClustering** class of cluster module of scikit learn library.

Then we have created the object of this class named as **hc.** The AgglomerativeClustering class takes the following parameters:

- **n_clusters=5**: It defines the number of clusters, and we have taken here 5 because it is the optimal number of clusters.
- **affinity='euclidean'**: It is a metric used to compute the linkage.
- **linkage='ward'**: It defines the linkage criteria, here we have used the "ward" linkage. This method is the popular linkage method that we have already used for creating the Dendrogram. It reduces the variance in each cluster.

In the last line, we have created the dependent variable y_pred to fit or train the model. It does train not only the model but also returns the clusters to which each data point belongs.

After executing the above lines of code, if we go through the variable explorer option in our Sypder IDE, we can check the y_pred variable. We can compare the original dataset with the y_pred variable. Consider the below image:



As we can see in the above image, the **y_pred** shows the clusters value, which means the customer id 1 belongs to the 5th cluster (as indexing starts from 0, so 4 means 5th cluster), the customer id 2 belongs to 4th cluster, and so on.

Step-4: Visualizing the clusters

As we have trained our model successfully, now we can visualize the clusters corresponding to the dataset.

Here we will use the same lines of code as we did in k-means clustering, except one change. Here we will not plot the centroid that we did in k-means, because here we have used dendrogram to determine the optimal number of clusters. The code is given below:

```
1. #visulaizing the clusters
2. mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
```

3. mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
4. mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
5. mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
6. mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
7. mtp.title('Clusters of customers')
8. mtp.xlabel('Annual Income (k$)')
9. mtp.ylabel('Spending Score (1-100)')
10. mtp.legend()
11. mtp.show()

Output: By executing the above lines of code, we will get the below output: