A Beginner's Guide to Hierarchical Clustering and how to Perform it in Python

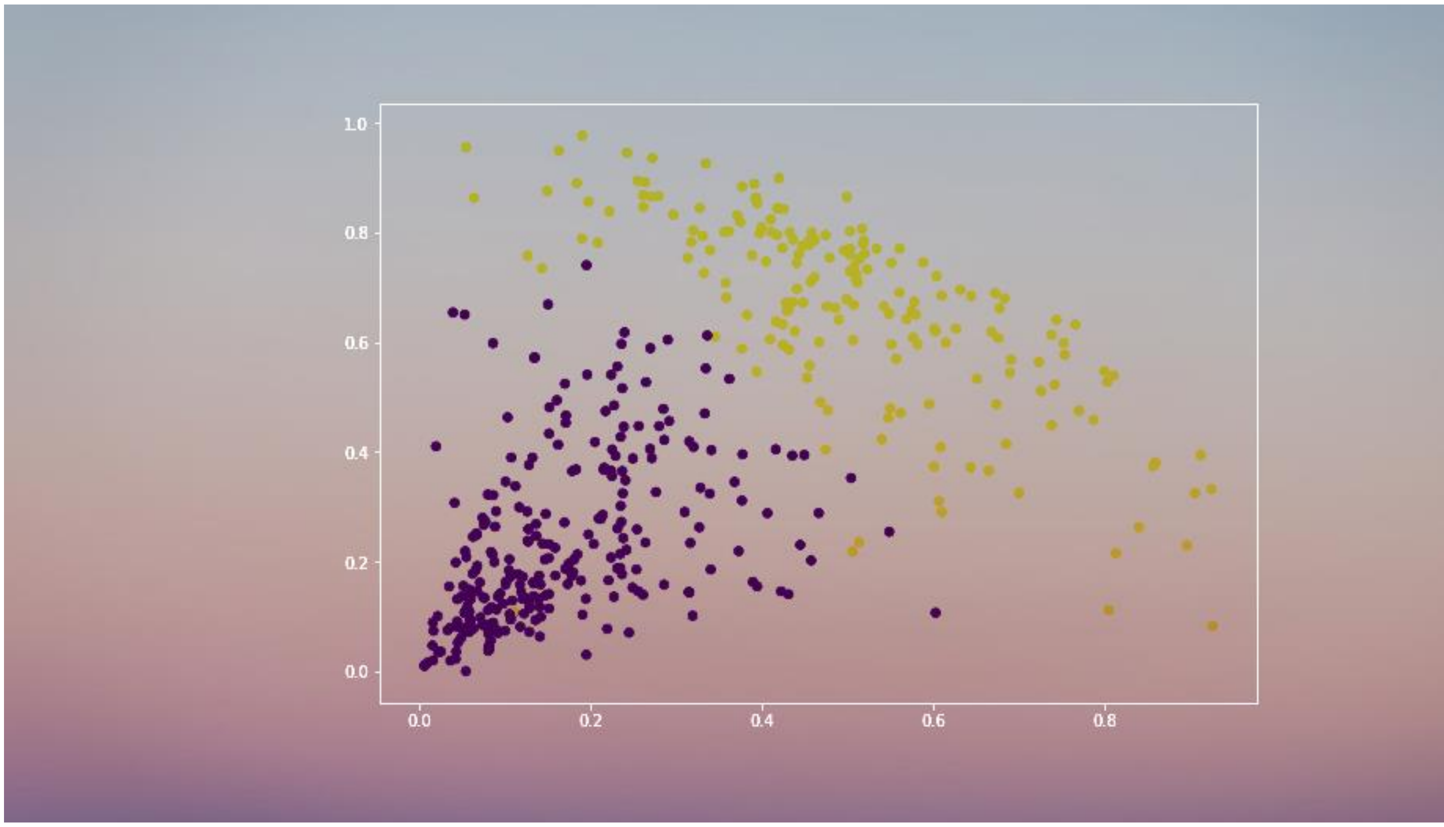
PULKIT SHARMA, MAY 27, 2019

Introduction

It is crucial to understand customer behavior in any industry. I realized this last year when my chief marketing officer asked me – "Can you tell me which existing customers should we target for our new product?"

That was quite a learning curve for me. I quickly realized as a data scientist how important it is to segment customers so my organization can tailor and build targeted strategies. This is where the concept of clustering came in ever so handy!

Problems like segmenting customers are often deceptively tricky because we are not working with any target variable in mind. We are officially in the land of unsupervised learning where we need to figure out patterns and structures without a set outcome in mind. It's both challenging and thrilling as a data scientist.



Now, there are a few different ways to perform clustering (as you'll see below). I will introduce you to one such type in this article – hierarchical clustering.

We will learn what hierarchical clustering is, its advantage over the other clustering algorithms, the different types of hierarchical clustering and the steps to perform it. We will finally take up a customer segmentation dataset and then implement hierarchical clustering in Python. I love this technique and I'm sure you will too after this article!

Note: As mentioned, there are multiple ways to perform clustering. I encourage you to check out our awesome guide to the different types of clustering:

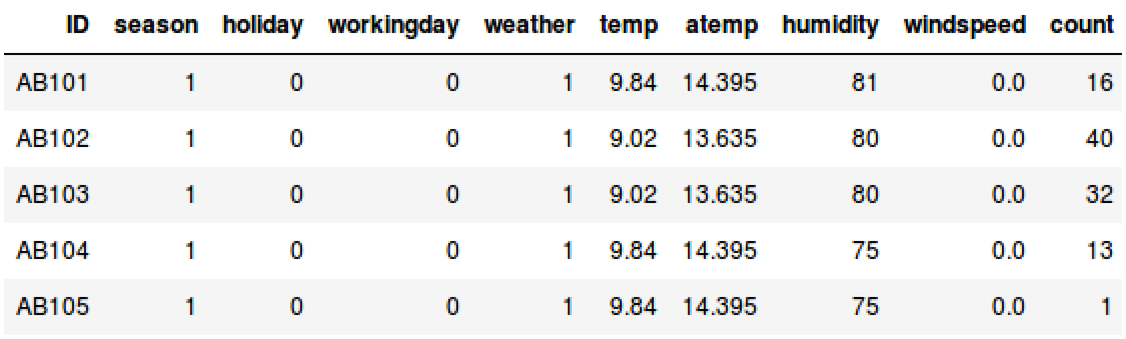
- *An Introduction to Clustering and different methods of clustering*

Table of Contents

Supervised vs Unsupervised Learning

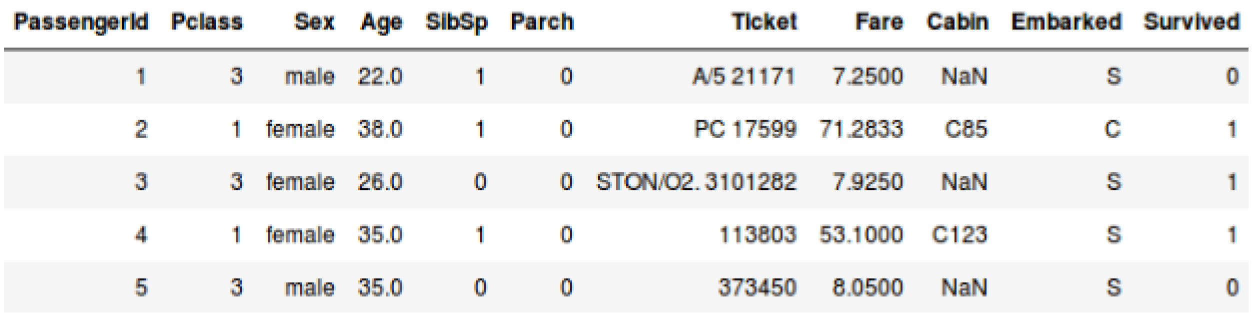
It’s important to understand the difference between supervised and unsupervised learningunsupervised learningbefore we dive into hierarchical clustering. Let me explain this difference using a simple example.

Suppose we want to estimate the count of bikes that will be rented in a city every day:



ID	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count
AB101	1	0	0	1	9.84	14.395	81	0.0	16
AB102	1	0	0	1	9.02	13.635	80	0.0	40
AB103	1	0	0	1	9.02	13.635	80	0.0	32
AB104	1	0	0	1	9.84	14.395	75	0.0	13
AB105	1	0	0	1	9.84	14.395	75	0.0	1

Or, let’s say we want to predict whether a person on board the Titanic survived or not:



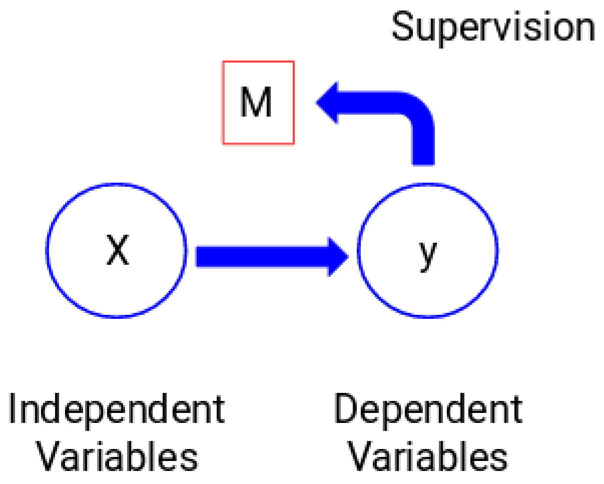
PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
1	3	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0
2	1	female	38.0	1	0	PC 17599	71.2833	C85	C	1
3	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	1
4	1	female	35.0	1	0	113803	53.1000	C123	S	1
5	3	male	35.0	0	0	373450	8.0500	NaN	S	0

We have a fixed target to achieve in both these examples:

- In the first example, we have to predict the count of bikes based on features like the season, holiday, workingday, weather, temp, etc.
- We are predicting whether a passenger survived or not in the second example. In the ‘Survived’ variable, 0 represents that the person did not survive and 1 means the person did make it out alive. The independent variables here include Pclass, Sex, Age, Fare, etc.

So, when we are given a target variable (count and Survival in the above two cases) which we have to predict based on a given set of predictors or independent variables (season, holiday, Sex, Age, etc.), such problems are called supervised learning problems.

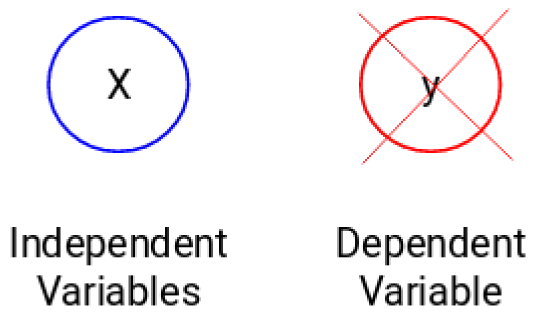
Let’s look at the figure below to understand this visually:



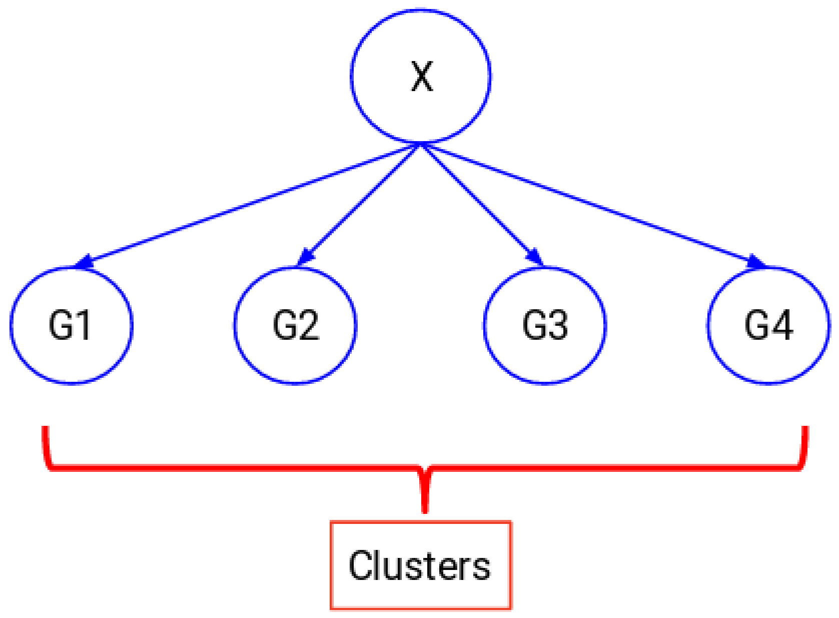
Here, y is our dependent or target variable, and X represents the independent variables. The target variable is dependent on X and hence it is also called a dependent variable. **We train our model using the independent variables in the supervision of the target variable and hence the name supervised learning.**

Our aim, when training the model, is to generate a function that maps the independent variables to the desired target. Once the model is trained, we can pass new sets of observations and the model will predict the target for them. This, in a nutshell, is supervised learning.

There might be situations when we do not have any target variable to predict. Such problems, without any explicit target variable, are known as unsupervised learning problems. We only have the independent variables and no target/dependent variable in these problems.



We try to divide the entire data into a set of groups in these cases. These groups are known as clusters and the process of making these clusters is known as **clustering**.



This technique is generally used for clustering a population into different groups. A few common examples include segmenting customers, clustering similar documents together, recommending similar songs or movies, etc.

There are a LOT more applications of unsupervised learning. If you come across any interesting application, feel free to share them in the comments section below!

Now, there are various algorithms that help us to make these clusters. The most commonly used clustering algorithms are K-means and Hierarchical clustering.

Why Hierarchical Clustering?

We should first know how K-means works before we dive into hierarchical clustering. Trust me, it will make the concept of hierarchical clustering all the more easier.

Here’s a brief overview of how K-means works:

1. Decide the number of clusters (k)
2. Select k random points from the data as centroids
3. Assign all the points to the nearest cluster centroid
4. Calculate the centroid of newly formed clusters
5. Repeat steps 3 and 4

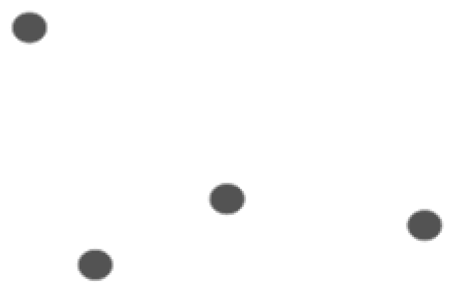
It is an iterative process. It will keep on running until the centroids of newly formed clusters do not change or the maximum number of iterations are reached.

But there are certain challenges with K-means. It always tries to make clusters of the same size. Also, we have to decide the number of clusters at the *beginning* of the algorithm. Ideally, we would not know how many clusters should we have, in the beginning of the algorithm and hence it a challenge with K-means.

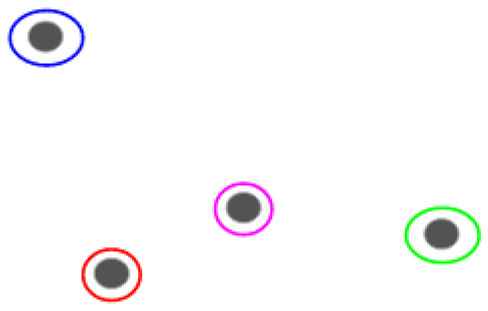
This is a gap hierarchical clustering bridges with aplomb. It takes away the problem of having to pre-define the number of clusters. Sounds like a dream! So, let’s see what hierarchical clustering is and how it improves on K-means.

What is Hierarchical Clustering?

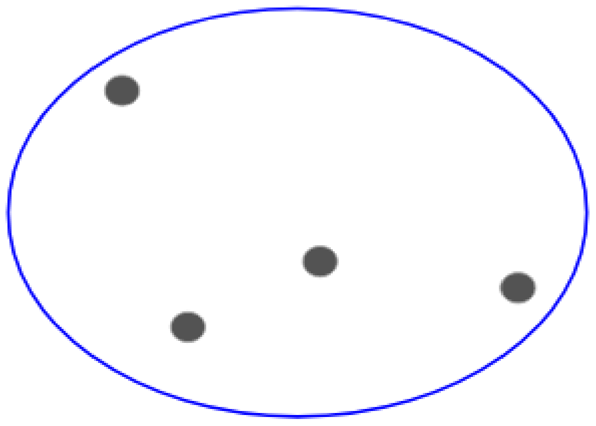
Let’s say we have the below points and we want to cluster them into groups:



We can assign each of these points to a separate cluster:



Now, based on the similarity of these clusters, we can combine the most similar clusters together and repeat this process until only a single cluster is left:



We are essentially building a hierarchy of clusters. That's why this algorithm is called hierarchical clustering. I will discuss how to decide the number of clusters in a later section. For now, let's look at the different types of hierarchical clustering.

Types of Hierarchical Clustering

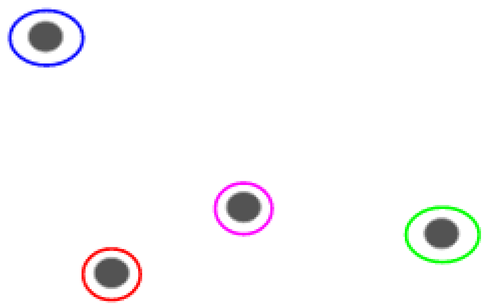
There are mainly two types of hierarchical clustering:

1. Agglomerative hierarchical clustering
2. Divisive Hierarchical clustering

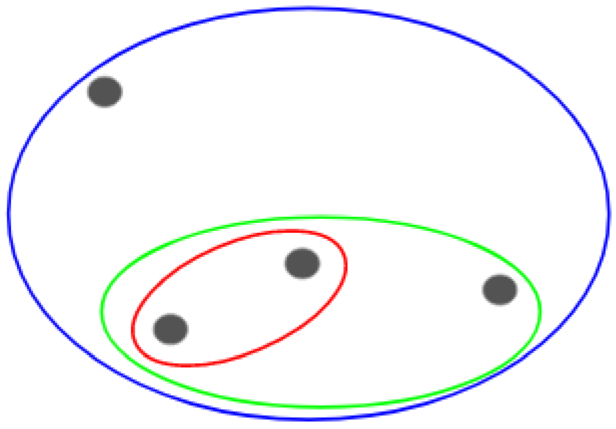
Let's understand each type in detail.

Agglomerative Hierarchical Clustering

We assign each point to an individual cluster in this technique. Suppose there are 4 data points. We will assign each of these points to a cluster and hence will have 4 clusters in the beginning:



Then, at each iteration, we merge the closest pair of clusters and repeat this step until only a single cluster is left:

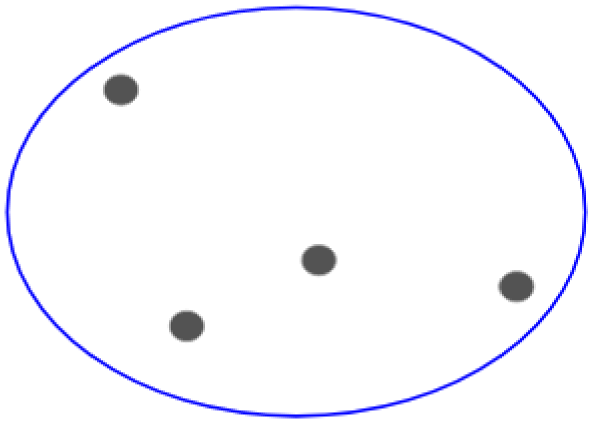


We are merging (or adding) the clusters at each step, right? Hence, this type of clustering is also known as **additive hierarchical clustering.**

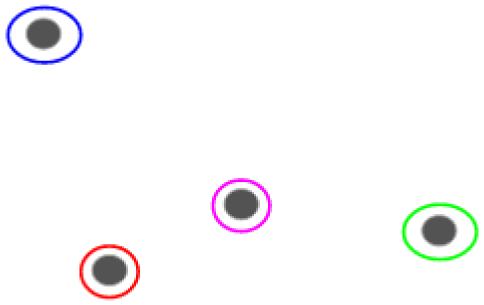
Divisive Hierarchical Clustering

Divisive hierarchical clustering works in the opposite way. Instead of starting with n clusters (in case of n observations), we start with a single cluster and assign all the points to that cluster.

So, it doesn't matter if we have 10 or 1000 data points. All these points will belong to the same cluster at the beginning:



Now, at each iteration, we split the farthest point in the cluster and repeat this process until each cluster only contains a single point:



We are splitting (or dividing) the clusters at each step, hence the name divisive hierarchical clustering.

Agglomerative Clustering is widely used in the industry and that will be the focus in this article. Divisive hierarchical clustering will be a piece of cake once we have a handle on the agglomerative type.

Steps to Perform Hierarchical Clustering

We merge the most similar points or clusters in hierarchical clustering – we know this. Now the question is – how do we decide which points are similar and which are not? It's one of the most important questions in clustering!

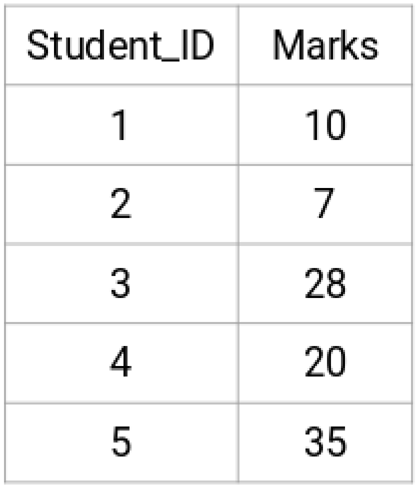
Here's one way to calculate similarity – Take the distance between the centroids of these clusters. The points having the least distance are referred to as similar points and we can merge them. We can refer to this as a **distance-based algorithm** as well (since we are calculating the distances between the clusters).

In hierarchical clustering, we have a concept called a **proximity matrix**. This stores the distances between each point. Let’s take an example to understand this matrix as well as the steps to perform hierarchical clustering.

Setting up the Example

Suppose a teacher wants to divide her students into different groups. She has the marks scored by each student in an assignment and based on these marks, she wants to segment them into groups. There’s no fixed target here as to how many groups to have. Since the teacher does not know what type of students should be assigned to which group, it cannot be solved as a supervised learning problem. So, we will try to apply hierarchical clustering here and segment the students into different groups.

Let’s take a sample of 5 students:

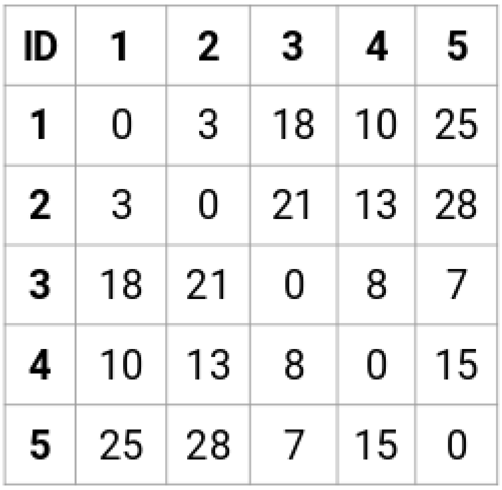


Student_ID	Marks
1	10
2	7
3	28
4	20
5	35

Creating a Proximity Matrix

First, we will create a proximity matrix which will tell us the distance between each of these points. Since we are calculating the distance of each point from each of the other points, we will get a square matrix of shape n X n (where n is the number of observations).

Let's make the 5 x 5 proximity matrix for our example:



ID	1	2	3	4	5
1	0	3	18	10	25
2	3	0	21	13	28
3	18	21	0	8	7
4	10	13	8	0	15
5	25	28	7	15	0

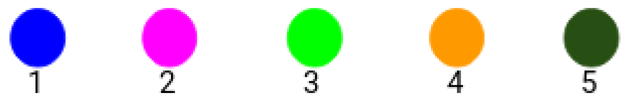
The diagonal elements of this matrix will always be 0 as the distance of a point with itself is always 0. We will use the Euclidean distance formula to calculate the rest of the distances. So, let's say we want to calculate the distance between point 1 and 2:

$$\sqrt{(10-7)^2} = \sqrt{9} = 3$$

Similarly, we can calculate all the distances and fill the proximity matrix.

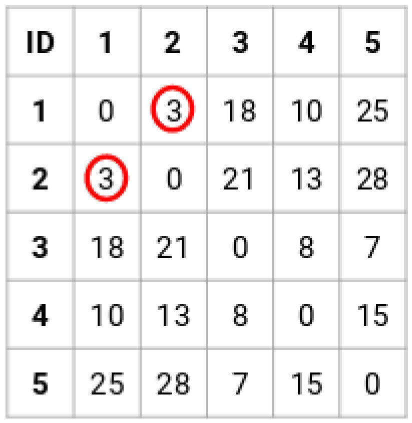
Steps to Perform Hierarchical Clustering

Step 1: First, we assign all the points to an individual cluster:



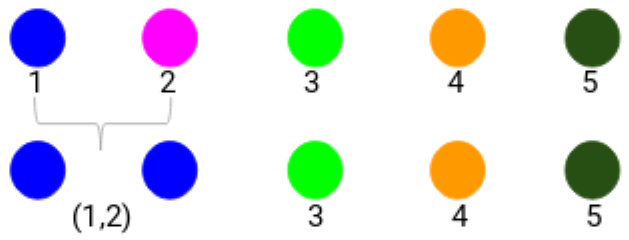
Different colors here represent different clusters. You can see that we have 5 different clusters for the 5 points in our data.

Step 2: Next, we will look at the smallest distance in the proximity matrix and merge the points with the smallest distance. We then update the proximity matrix:

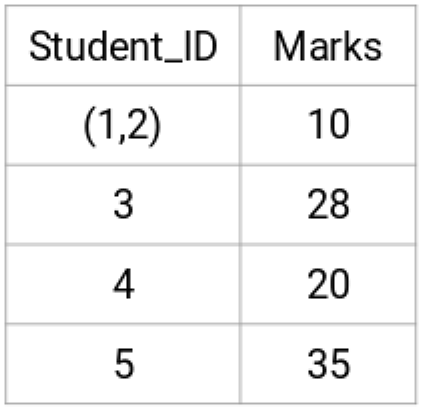


ID	1	2	3	4	5
1	0	3	18	10	25
2	3	0	21	13	28
3	18	21	0	8	7
4	10	13	8	0	15
5	25	28	7	15	0

Here, the smallest distance is 3 and hence we will merge point 1 and 2:

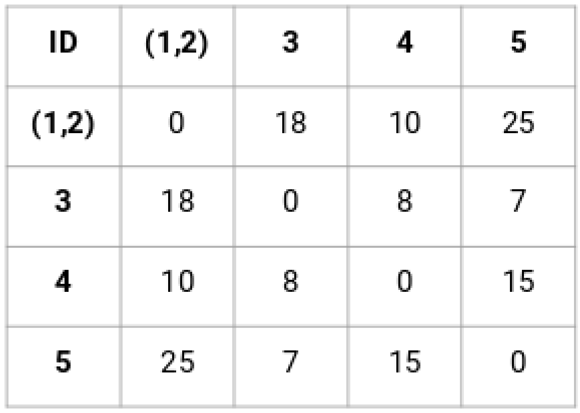


Let’s look at the updated clusters and accordingly update the proximity matrix:



Student_ID	Marks
(1,2)	10
3	28
4	20
5	35

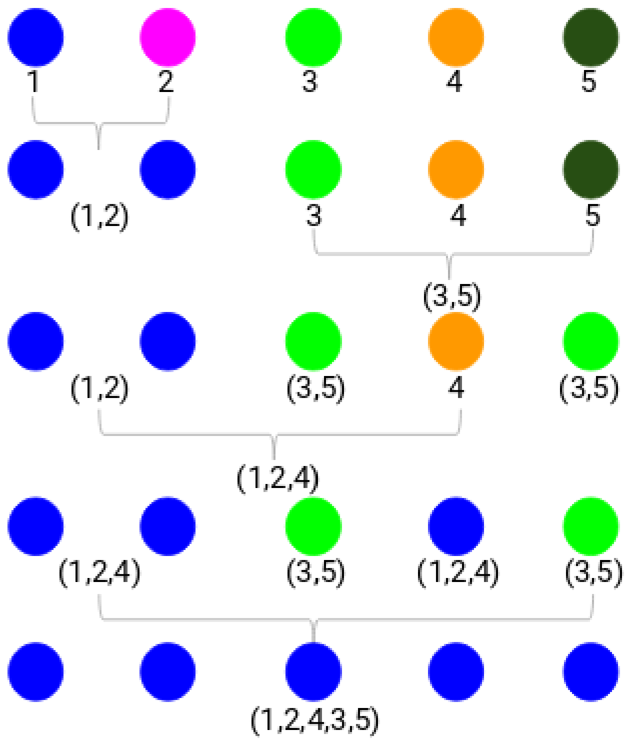
Here, we have taken the maximum of the two marks (7, 10) to replace the marks for this cluster. Instead of the maximum, we can also take the minimum value or the average values as well. Now, we will again calculate the proximity matrix for these clusters:



ID	**(1,2)**	**3**	**4**	**5**
(1,2)	0	18	10	25
3	18	0	8	7
4	10	8	0	15
5	25	7	15	0

Step 3: We will repeat step 2 until only a single cluster is left.

So, we will first look at the minimum distance in the proximity matrix and then merge the closest pair of clusters. We will get the merged clusters as shown below after repeating these steps:



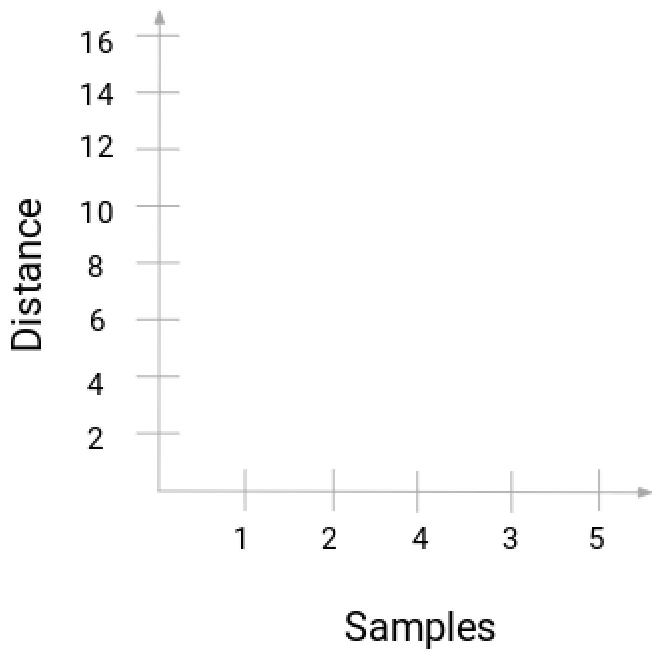
We started with 5 clusters and finally have a single cluster. **This is how agglomerative hierarchical clustering works**. But the burning question still remains – how do we decide the number of clusters? Let's understand that in the next section.

How should we Choose the Number of Clusters in Hierarchical Clustering?

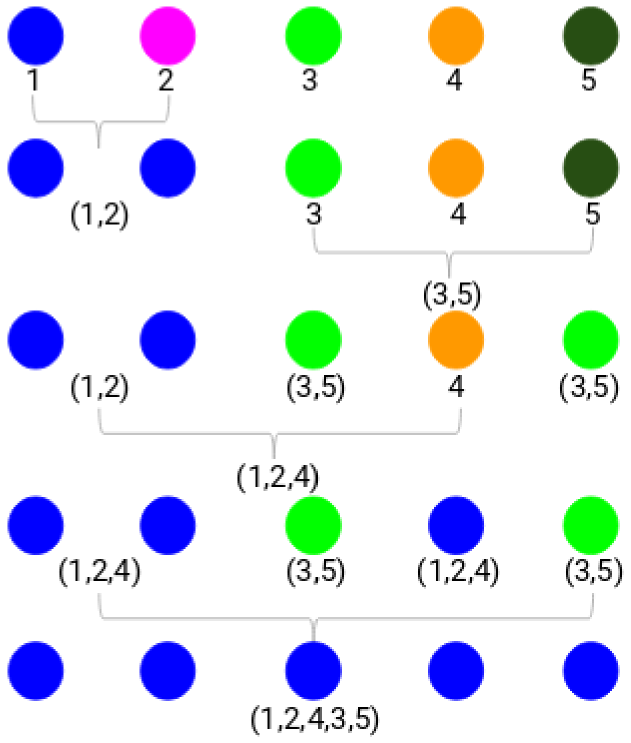
Ready to finally answer this question that's been hanging around since we started learning? To get the number of clusters for hierarchical clustering, we make use of an awesome concept called a **Dendrogram**.

A dendrogram is a tree-like diagram that records the sequences of merges or splits.

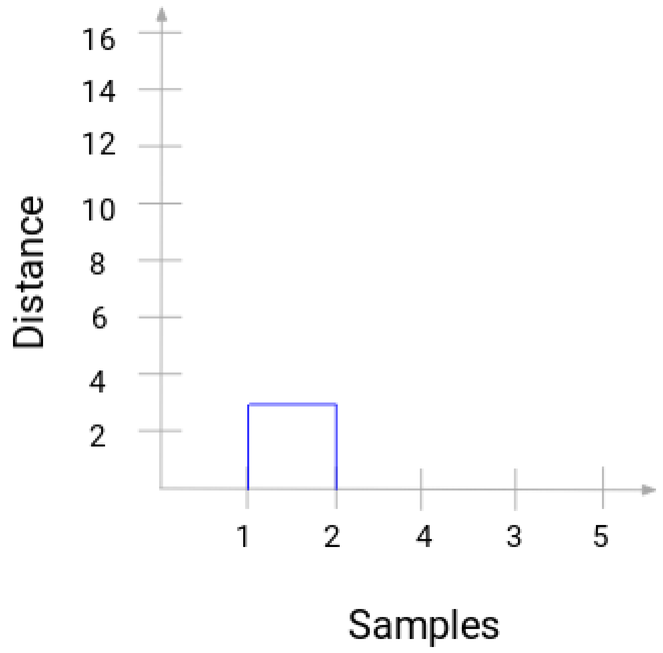
Let's get back to our teacher-student example. Whenever we merge two clusters, a dendrogram will record the distance between these clusters and represent it in graph form. Let's see how a dendrogram looks like:



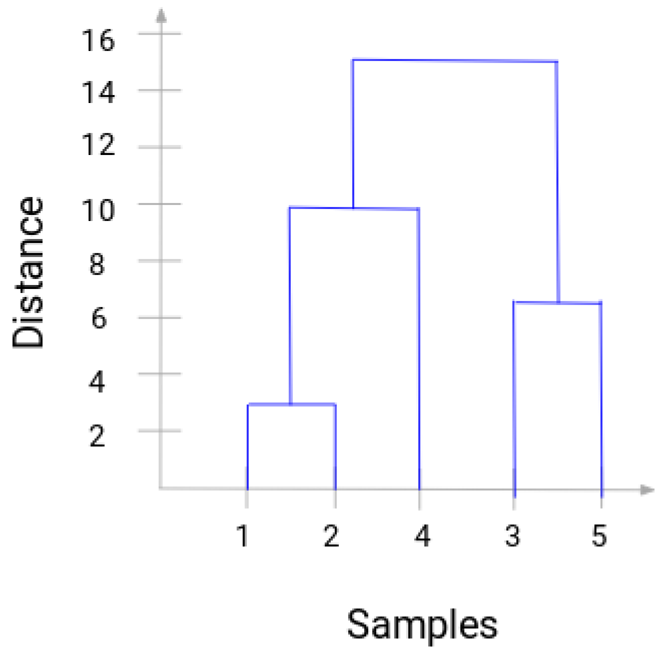
We have the samples of the dataset on the x-axis and the distance on the y-axis. **Whenever two clusters are merged, we will join them in this dendrogram and the height of the join will be the distance between these points.** Let's build the dendrogram for our example:



Take a moment to process the above image. We started by merging sample 1 and 2 and the distance between these two samples was 3 (refer to the first proximity matrix in the previous section). Let’s plot this in the dendrogram:

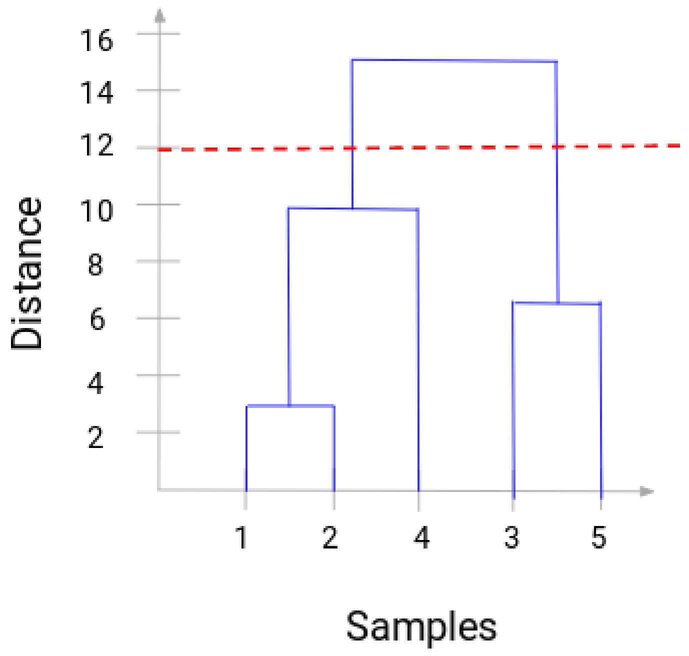


Here, we can see that we have merged sample 1 and 2. The vertical line represents the distance between these samples. Similarly, we plot all the steps where we merged the clusters and finally, we get a dendrogram like this:



We can clearly visualize the steps of hierarchical clustering. **More the distance of the vertical lines in the dendrogram, more the distance between those clusters.**

Now, we can set a threshold distance and draw a horizontal line (*Generally, we try to set the threshold in such a way that it cuts the tallest vertical line*). Let's set this threshold as 12 and draw a horizontal line:



The number of clusters will be the number of vertical lines which are being intersected by the line drawn using the threshold. In the above example, since the red line intersects 2 vertical lines, we will have 2 clusters. One cluster will have a sample (1,2,4) and the other will have a sample (3,5). Pretty straightforward, right?

This is how we can decide the number of clusters using a dendrogram in Hierarchical Clustering. In the next section, we will implement hierarchical clustering which will help you to understand all the concepts that we have learned in this article.

Solving the Wholesale Customer Segmentation problem using Hierarchical Clustering

Time to get our hands dirty in Python!

We will be working on a wholesale customer segmentation problem. You can download the dataset using **this link**. The data is hosted on the UCI Machine Learning repository. The aim of this problem is to segment the clients of a wholesale distributor based on their annual spending on diverse product categories, like milk, grocery, region, etc.

Let’s explore the data first and then apply Hierarchical Clustering to segment the clients.

We will first import the required libraries:

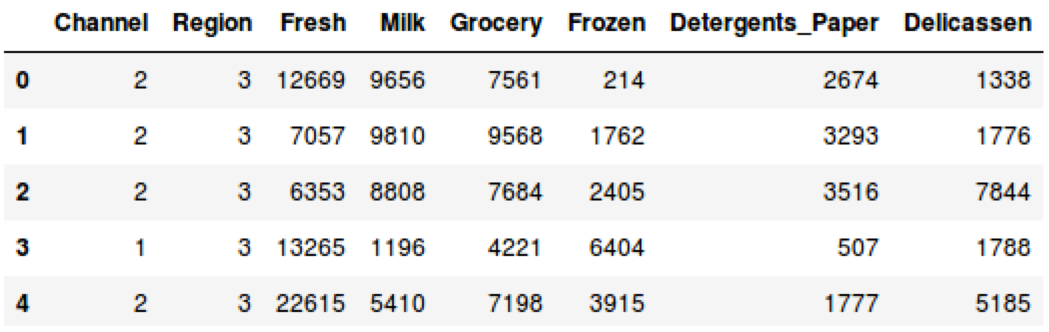
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

view rawimporting_libraries.py hosted with by GitHub

Load the data and look at the first few rows:

```
data = pd.read_csv('Wholesale customers data.csv')
data.head()
```

view rawreading_data.py hosted with by GitHub



	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

There are multiple product categories – Fresh, Milk, Grocery, etc. The values represent the number of units purchased by each client for each produc**t. Our aim is to make clusters from this data that can segment similar clients together**. We will, of course, use Hierarchical Clustering for this problem.

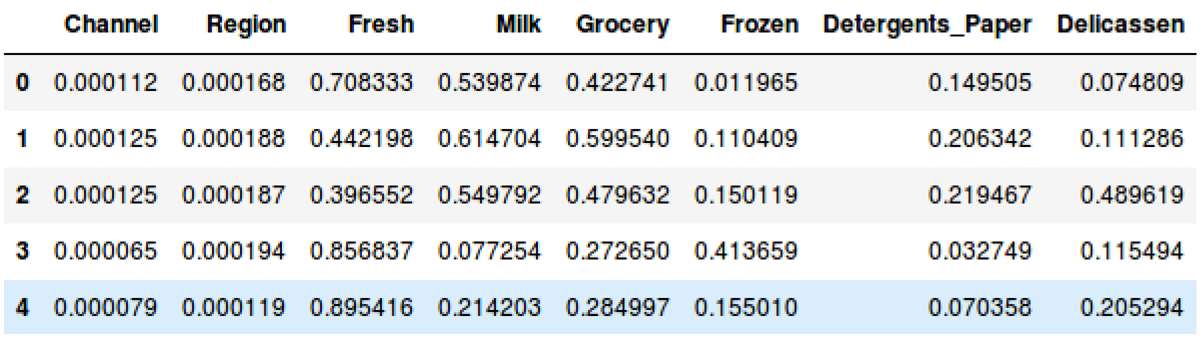
But before applying Hierarchical Clustering, we have to normalize the data so that the scale of each variable is the same. Why is this important? Well, if the scale of the variables is not the same, the model might become biased towards the variables with a higher magnitude like Fresh or Milk (refer to the above table).

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()
```

view rawscaling_data.py hosted with by GitHub

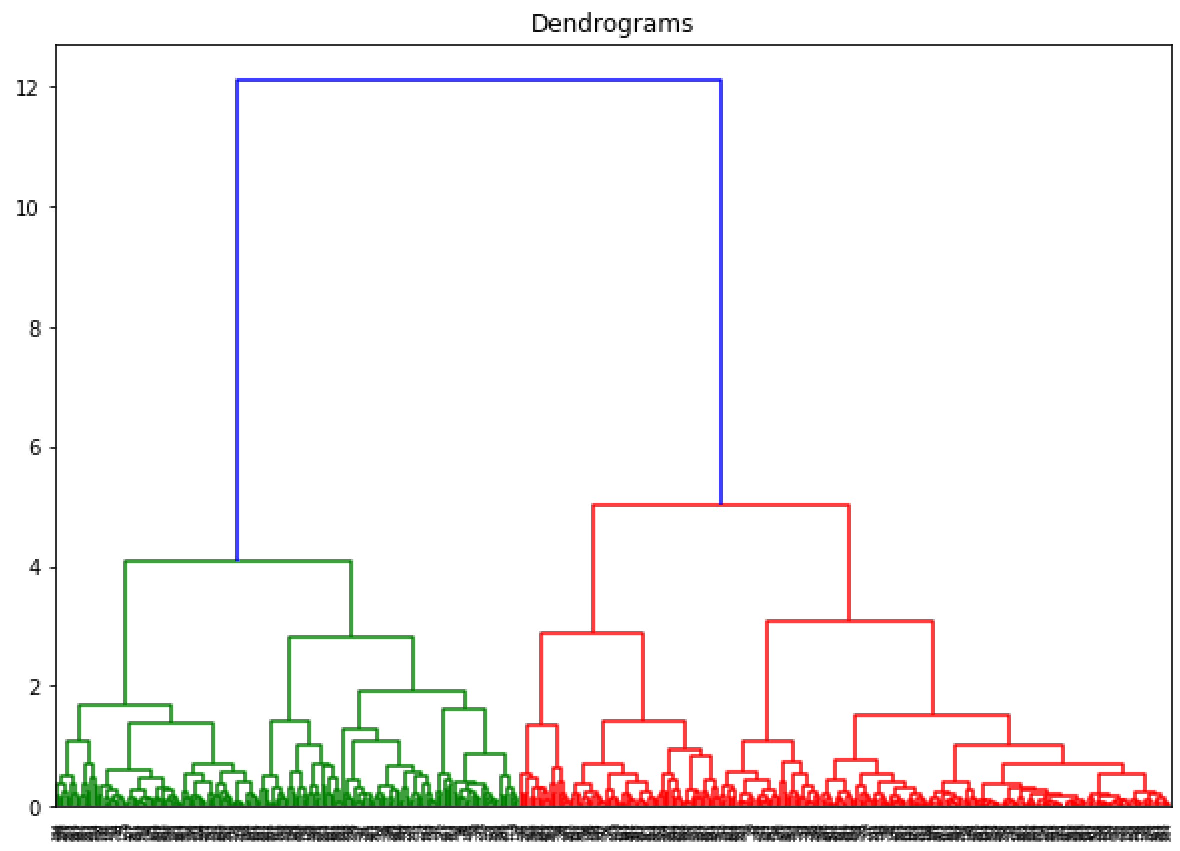


	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	0.000112	0.000168	0.708333	0.539874	0.422741	0.011965	0.149505	0.074809
1	0.000125	0.000188	0.442198	0.614704	0.599540	0.110409	0.206342	0.111286
2	0.000125	0.000187	0.396552	0.549792	0.479632	0.150119	0.219467	0.489619
3	0.000065	0.000194	0.856837	0.077254	0.272650	0.413659	0.032749	0.115494
4	0.000079	0.000119	0.895416	0.214203	0.284997	0.155010	0.070358	0.205294

Here, we can see that the scale of all the variables is almost similar. Now, we are good to go. Let's first draw the dendrogram to help us decide the number of clusters for this particular problem:

```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
```

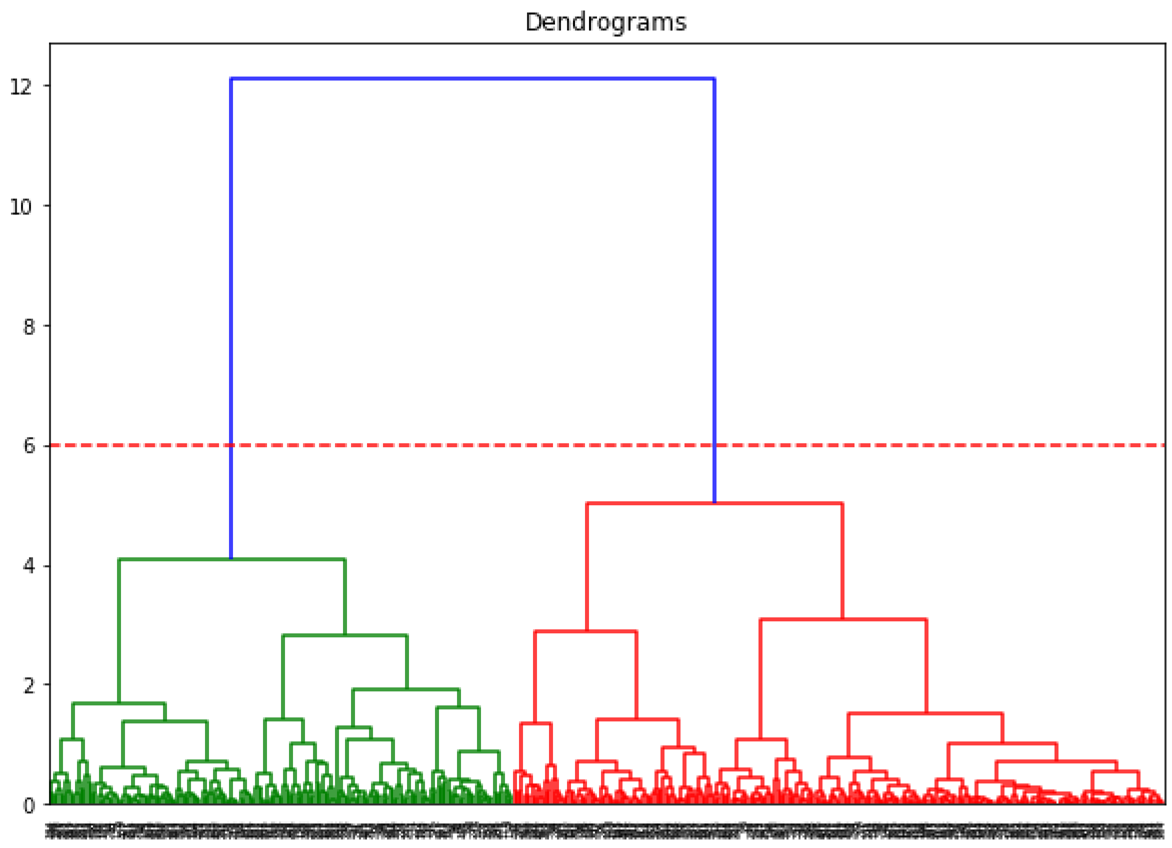
view rawdendrogram.py hosted with by GitHub



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:

```
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
plt.axhline(y=6, color='r', linestyle='--')
```

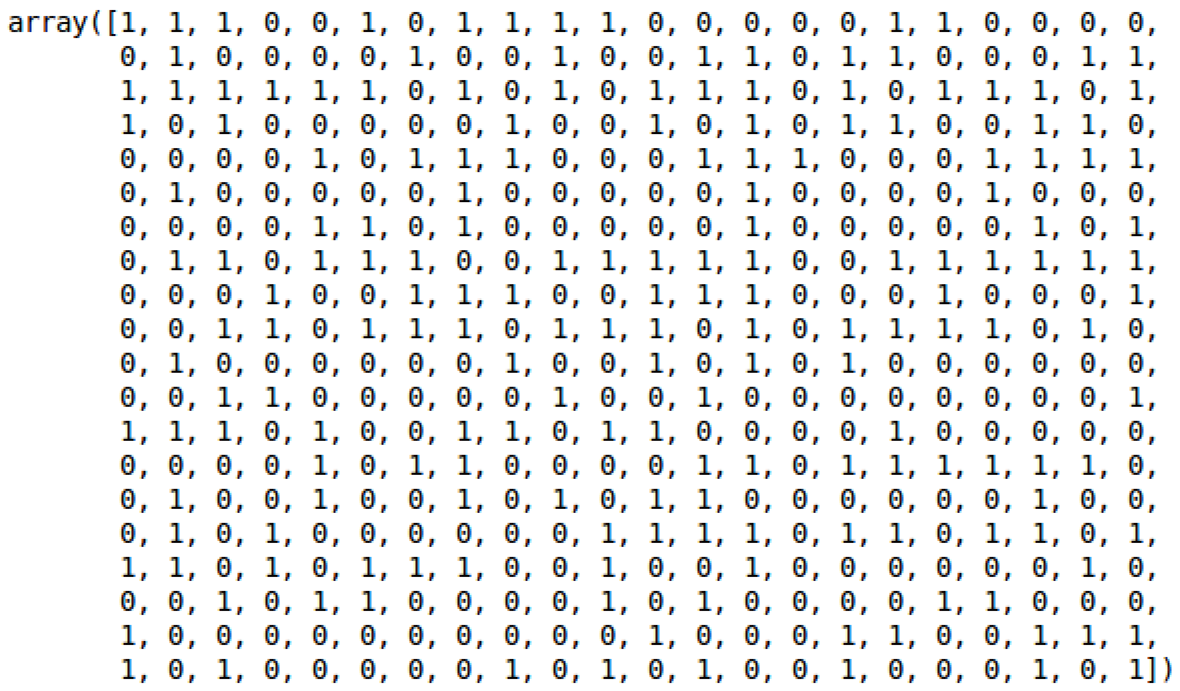
view raw **dendrogram_threshold.py** hosted with by **GitHub**



We have two clusters as this line cuts the dendrogram at two points. Let's now apply hierarchical clustering for 2 clusters:

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
cluster.fit_predict(data_scaled)
```

view rawhierarchical_clustering.py hosted with by **GitHub**

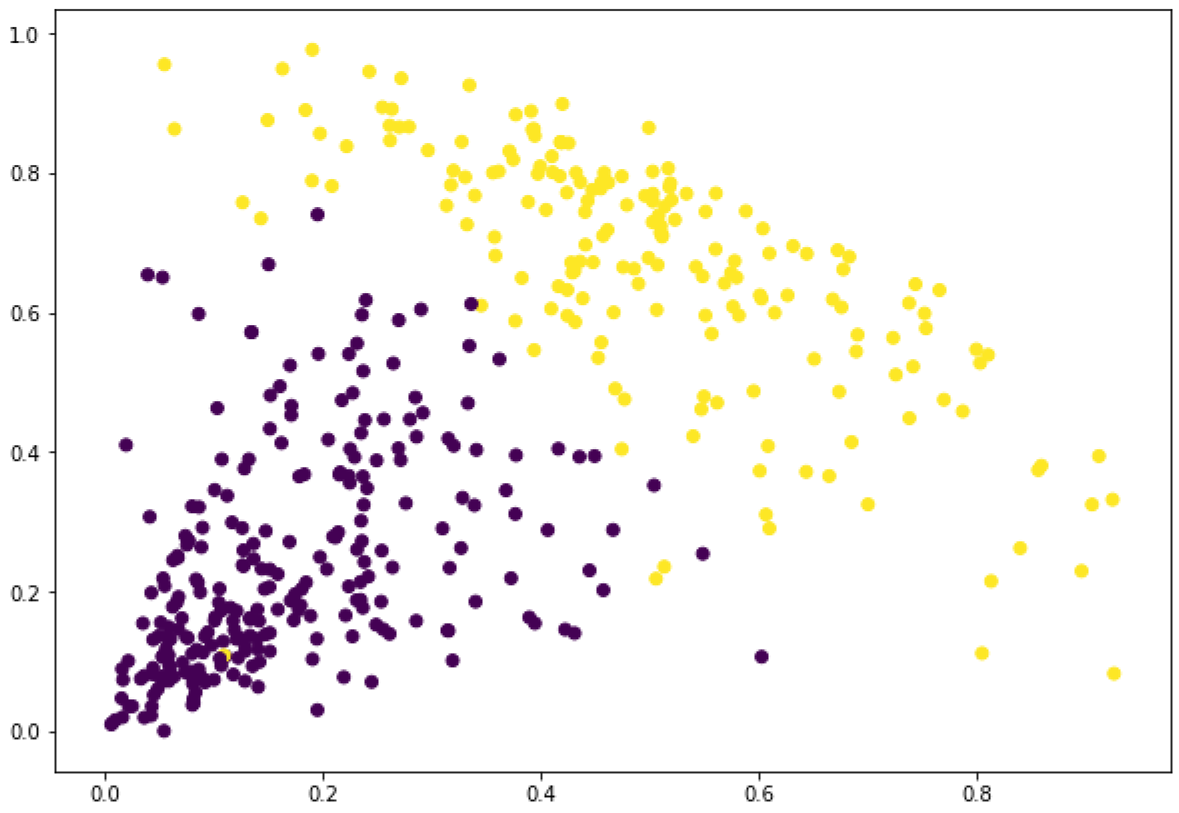


```
array([1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1])
```

We can see the values of 0s and 1s in the output since we defined 2 clusters. 0 represents the points that belong to the first cluster and 1 represents points in the second cluster. Let's now visualize the two clusters:

```python
plt.figure(figsize=(10, 7))
plt.scatter(data_scaled['Milk'], data_scaled['Grocery'], c=cluster.labels_)
```

view raw **clustering_visualization.py** hosted with by **GitHub**



Awesome! We can clearly visualize the two clusters here. This is how we can implement hierarchical clustering in Python.

End Notes

Hierarchical clustering is a super useful way of segmenting observations. The advantage of not having to pre-define the number of clusters gives it quite an edge over k-Means.

If you are still relatively new to data science, I highly recommend taking the **Applied Machine Learning** course. It is one of the most comprehensive end-to-end machine learning courses you will find anywhere. Hierarchical clustering is just one of a diverse range of topics we cover in the course.

What are your thoughts on hierarchical clustering? Do you feel there's a better way to create clusters using less computational resources? Connect with me in the comments section below and let's discuss