**Agglomerative Hierarchical Clustering**

**Hierarchical vs K-Means clustering**

When it comes to clustering tasks, most are undecided on two main methods: Hierarchical clustering or [K-means clustering](https://www.datacamp.com/tutorial/k-means-clustering-python).

But what makes them different? Let’s look at their differences below for a clearer picture.

* **K-Means**:
  + Requires the number of clusters to be defined before running.
  + Efficient and scalable for large datasets.
  + Assumes clusters are spherical and equally sized.
* **Hierarchical Clustering**:
  + Does not require a predefined number of clusters.
  + More interpretable due to dendrograms.
  + Slower on large datasets but better for exploring structure.

When comparing the two, some areas make hierarchical clustering a stand-out option.

Here are some areas where hierarchical clustering is good:

* Flexibility in cluster selection using dendrograms.
* Good for small datasets or when a visual hierarchy is needed.
* Handles complex cluster shapes.

However, it can have some downsides, such as:

* Computationally expensive
* Sensitive to noisy data and outliers.

You might ask: When should I use hierarchical clustering?

Hierarchical clustering is an excellent choice in the following applications:

* [Exploratory data analysis](https://www.datacamp.com/code-along/exploratory-data-analysis-python-beginners)
* Gene expression pattern analysis
* Market segmentation with moderate-sized datasets

**Introduction**

Clustering is an example of unsupervised learning, in which no training samples are available from which to learn and create model. Fast and robust clustering algorithms play an important role in extracting useful information in large databases.

The aim of cluster analysis is to partition a set of N object into C clusters such that objects within cluster should be similar to each other and objects in different clusters should be dissimilar with each other.

Clustering is a popular algorithm in a variety of fields, including psychology, business and retail, computational biology, social media network analysis, and so on.

In data mining, hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

Agglomerative: This is a “bottom up” approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. Divisive: This is a “top down” approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

This article explains the idea of agglomerative clustering, the steps to perform it, and finally takes a customer segmentation dataset to cluster clients of a wholesale distributor based on their annual spending on diverse product categories.

**Hierarchical vs K-Means Clustering Algorithms**

The advantage of hierarchical clustering over k-means clustering is that it’s easy to implement and outputs a hierarchy, that is, a structure that is more informative than the unstructured sets of flat clusters returned by k-means. Therefore, it is easier to decide on the number of clusters by looking at the dendogram.

Moreover, the advantage of not having to predefine the number of clusters gives it quite an edge over k-Means. However, hierarchical clustering is not suitable for large data sets due to its time complexity, and is also sensitive to outliers.

**Agglomerative Hierarchical Clustering**

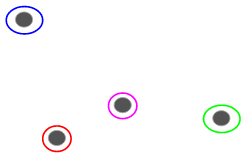
In Clustering, one of the most widely used algorithms is agglomerative algorithms. A bottom-up method in which each entity represents its own cluster, which is then iteratively merged until the desired cluster structure is achieved.

This N-sample algorithm starts with N clusters, each containing a single sample. Following that, two clusters with the greatest similarity will combine until the number of clusters is reduced to one or the user specifies.

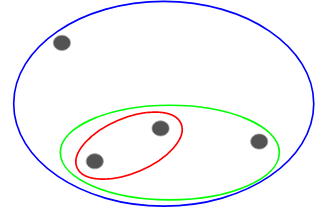
The steps for forming agglomerative (bottom-up) clustering are:

1. Start by considering each data point as its own singleton cluster.
2. After each iteration of calculating Euclidian distance, merge two clusters with minimum distance.
3. Stop when there is a single cluster of all examples, else go to step 2

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:



**Customer Segmentation using Hierarchical Clustering in Python**

We will be working on a wholesale customer segmentation problem. 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, etc.

**Data Description**

Wholesale customer dataset consists of 440 instances and eight (8) attributes that include Channel, Region, Fresh, Milk, Grocery, Frozen, Detergents\_paper and Delicassen.

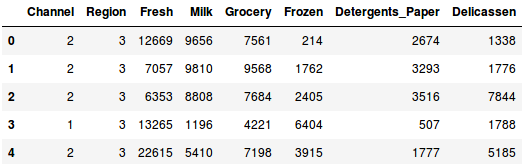
**Data Exploration**

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 matplotlib.pyplot as plt

Load the data and look at the first few rows:

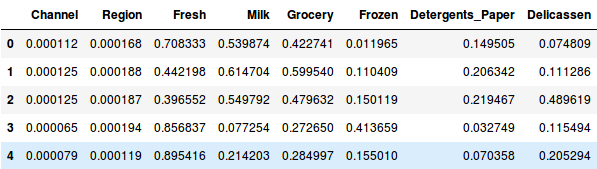
data = pd.read\_csv('data/Wholesale customers data.csv')  
data.head()



**Data Normalization**

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()

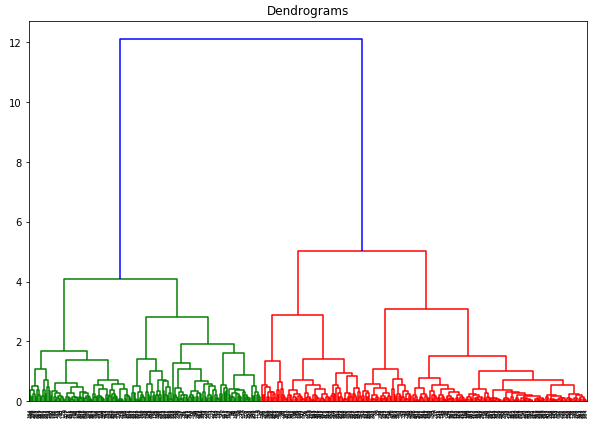


Here, we can see that the scale of all the variables is almost similar. Now, we are good to go. Let’s first draw the dendrogram to help us decide the number of clusters for this particular problem.

**Dendogram**

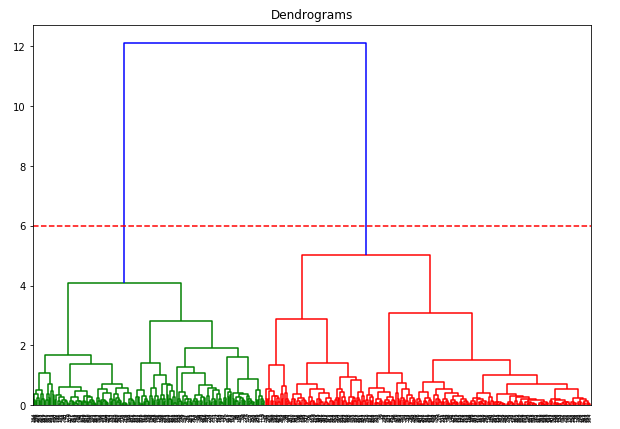
Dendogram is a tree-like diagram that records the sequence of merges or splits. To get the number of clusters for hierarchical clustering we use the concept of dendogram.

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



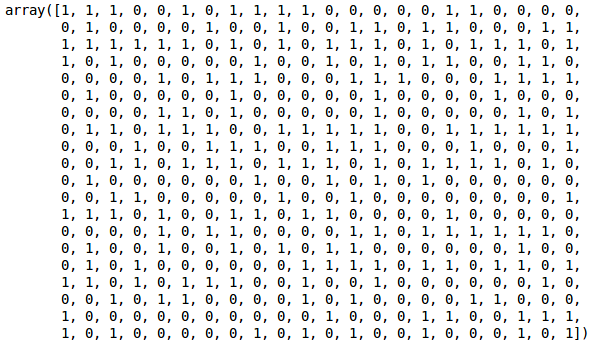
The x-axis contains the samples and y-axis represents the distance between these samples. The vertical line with maximum distance is the blue line and hence we can decide a threshold of 6 and cut the dendrogram:

plt.title("Dendrograms")  
dend = shc.dendrogram(shc.linkage(data\_scaled, method='ward'))  
plt.axhline(y=6, color='r', linestyle='--')



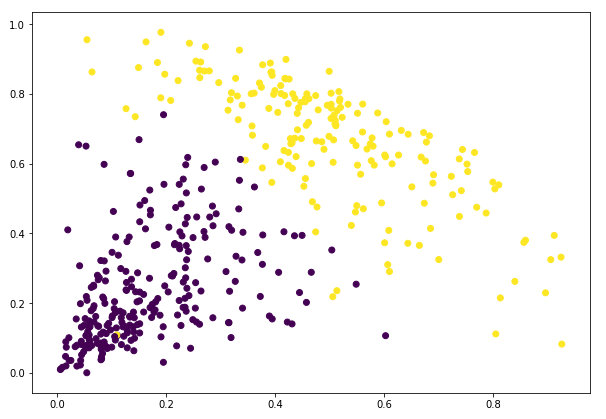
We have two clusters, as this line cuts the dendrogram at two points. Let’s now apply hierarchical clustering for 2 clusters:

from sklearn.cluster import AgglomerativeClustering  
cluster = AgglomerativeClustering(n\_clusters=2, affinity='euclidean', linkage='ward')   
cluster.fit\_predict(data\_scaled)



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:

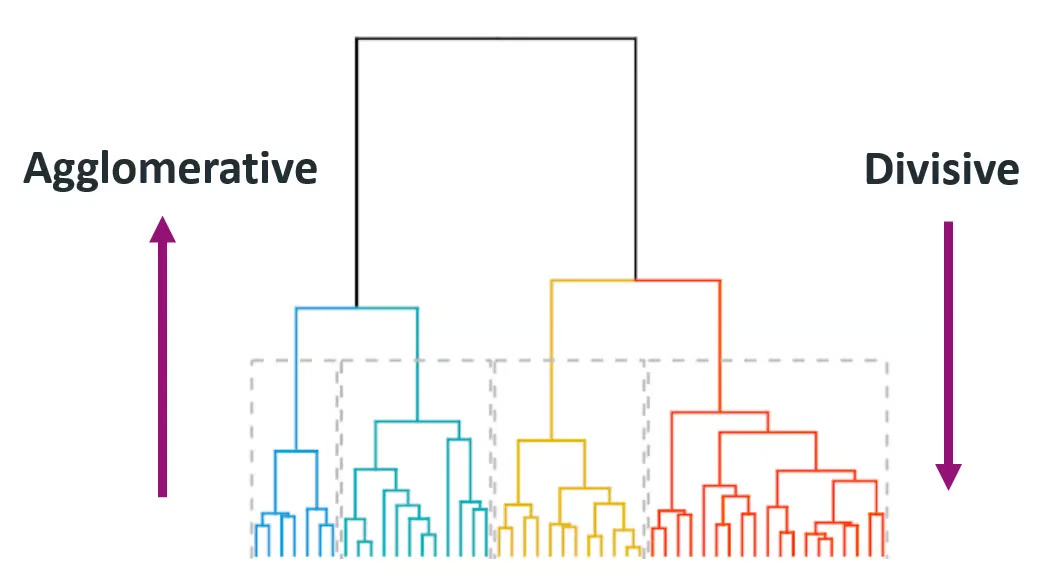
plt.scatter(data\_scaled['Milk'], data\_scaled['Grocery'], c=cluster.labels\_)

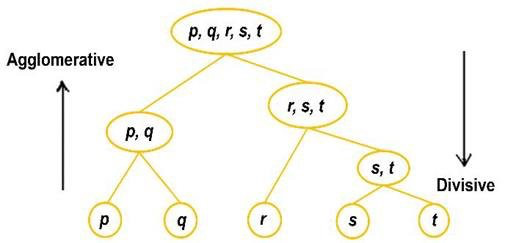


Great! We can clearly visualize the two clusters here.

**Types of Hierarchical Clustering**

Within the technique of hierarchical clustering, you’ll expect several types, with each providing different insights and results. The two main types of hierarchical clustering are agglomerative (bottom-up) and divisive (top-down).





*Source:* [*Himanshu Sharma on Medium*](https://harshsharma1091996.medium.com/hierarchical-clustering-996745fe656b)

**1. Agglomerative (Bottom-Up)**

Agglomerative clustering starts with each data point as an individual cluster and iteratively merges the closest pair of clusters until only one cluster remains or until a stopping condition is met (like a desired number of clusters).

This method is also called bottom-up because it starts from the bottom (individual data points) and builds up to the top (final cluster).

**Step-by-step guide**:

1. Treat each data point as a singleton cluster.
2. Compute the pairwise distances between all clusters.
3. Merge the two clusters with the smallest distance.
4. Update the distance matrix to reflect the new set of clusters.
5. Repeat steps 2–4 until all points belong to a single cluster.

This method is widely used due to its simplicity and ease of implementation.

Here’s an example of how this can be implemented in Python:

import numpy as np

from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

import matplotlib.pyplot as plt

# Samples data

data = np.array([[1, 2], [2, 3], [5, 8], [8, 8], [1, 0], [0, 1]])

# Applies agglomerative clustering using Ward's method

Z = linkage(data, method='ward')

# Plots dendrogram

plt.figure(figsize=(8, 4))

dendrogram(Z)

plt.title('Agglomerative Clustering Dendrogram')

plt.xlabel('Data Points')

plt.ylabel('Distance')

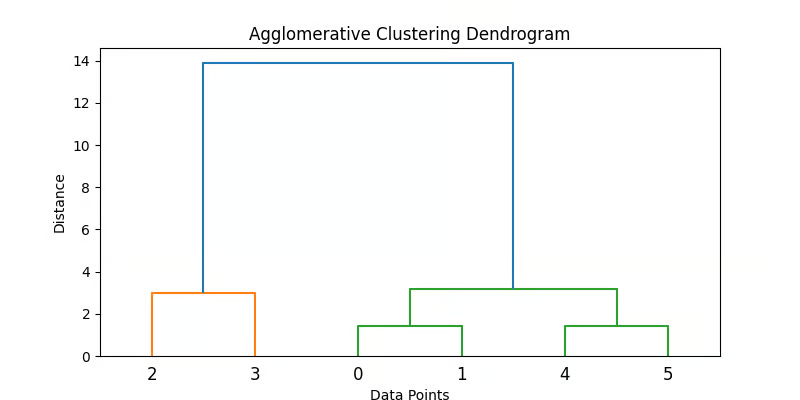
plt.show()

# Extracts clusters (e.g., form 2 clusters)

clusters = fcluster(Z, t=2, criterion='maxclust')

print("Cluster assignments:", clusters)

[**Powered By**](https://www.datacamp.com/datalab)



**2. Divisive (Top-Down)**

Divisive clustering starts with all data points in a single cluster and recursively splits them into smaller clusters. The process continues until each data point is in its own individual cluster.

It is also known as a top-down approach, as it starts at the top (single cluster) and breaks it down into smaller clusters.

**Step-by-step guide**:

1. Start with one large cluster containing all data points.
2. Split the cluster into two that are as different as possible.
3. Reapply the splitting process recursively to each resulting cluster.

Divisive methods are typically more computationally expensive due to their recursive nature, and accuracy depends greatly on the splitting strategy. Agglomerative methods are more common due to ease of implementation and widespread software support.

**Note:** Divisive hierarchical clustering is not as readily supported in standard Python libraries as agglomerative clustering. One approach is to use clustering algorithms like k-means recursively.

Here’s a code concept of a simulated recursive k-means splitting:

from sklearn.cluster import KMeans

def divisive\_clustering(data, depth=2):

if depth == 0 or len(data) <= 1:

return [data]

kmeans = KMeans(n\_clusters=2, random\_state=42).fit(data)

labels = kmeans.labels\_

cluster1 = data[labels == 0]

cluster2 = data[labels == 1]

return divisive\_clustering(cluster1, depth - 1) + divisive\_clustering(cluster2, depth - 1)

# Run recursive splitting to simulate divisive clustering

split\_clusters = divisive\_clustering(data, depth=2)

for i, cluster in enumerate(split\_clusters):

print(f"Cluster {i+1} size: {len(cluster)}")

[**Powered By**](https://www.datacamp.com/datalab)

This simplified code illustrates a conceptual approach to divisive clustering using recursive K-means. Note, however, that standard divisive methods like DIANA use different splitting criteria.

**Hierarchical Clustering vs KMeans**

The difference between Kmeans and hierarchical clustering is that in Kmeans clustering, the number of clusters is pre-defined and is denoted by “K”, but in hierarchical clustering, the number of sets is either one or similar to the number of data observations.

Another difference between these two clustering techniques is that K-means clustering is more effective on much larger datasets than hierarchical clustering. But hierarchical clustering spheroidal shape small datasets.

K-means clustering is effective on dataset spheroidal shape of clusters compared to hierarchical clustering.

Advantages

**1. Performance:**

It is effective in data observation from the data shape and returns accurate results. Unlike KMeans clustering, here, better performance is not limited to the spheroidal shape of the data; data having any values is acceptable for hierarchical clustering.

**2. Easy:**

It is easy to use and provides better user guidance with good community support. So much content and good documentation are available for a better user experience.

**3. More Approaches:**

Two approaches are there using which datasets can be trained and tested, agglomerative and divisive. So if the dataset provided is complex and very hard to train on, we can use another approach.

**4. Performance on Small Datasets:**

The hierarchical clustering algorithms are effective on small datasets and return accurate and reliable results with lower training and testing time.

Disadvantages

**1. Time Complexity:**

As many iterations and calculations are associated, the time complexity of hierarchical clustering is high. In some cases, it is one of the main reasons for preferring KMeans clustering.

**2. Space Complexity:**

As many calculations of errors with losses are associated with every epoch, the space complexity of the algorithm is very high. Due to this, while implementing the hierarchical clustering, the space of the model is considered. In such cases, we prefer KMeans clustering.

**3. Poor performance on Large Datasets:**

When training a hierarchical clustering algorithm for large datasets, the training process takes so much time with space which results in poor performance of the algorithms.

**Agglomerative Clustering**

Each dataset is one particular data observation and a set in agglomeration clustering. Based on the distance between groups, similar collections are merged based on the loss of the algorithm after one iteration. Again the loss value is calculated in the next iteration, where similar clusters are combined again. The process continues until we reach the minimum value of the loss.

**Code**

import pandas as pd

from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n\_clusters=5,affinity = 'l1',linkage='single')

data=pd.read\_csv('toy\_dataset.csv')

data.drop(columns=['Illness','City','Gender'],inplace=True)

cluster.fit\_predict(data)

Divisive Clustering

Divisive clustering is the opposite of agglomeration clustering. The whole dataset is considered a single set, and the loss is calculated. According to the Euclidian distance and similarity between data observations in the next iteration, the whole single set is divided into multiple clusters, hence the name “divisive.” This same process continues until we achieve the minimum loss value.

There is no method of implementing divisive clustering in Sklearn, although we can do it manually using the code below:

**Importing Required Libraries**

import numpy

import pandas

import copy

import matplotlib.pyplot

from ditsance\_matrix import distanceMatric

**Creating The Diana Class**

Class DianakClustering:

def \_\_init\_\_(self,datak):

self.data = datak

self.n\_samples, self.n\_features = datak.shape

def fit(self,no\_clusters):

self.n\_samples, self.n\_features = data.shape

similarity\_matrix = DistanceMatrix(self.datak)

clusters = [list(range(self.n\_samples))]

while True:

csd= [np.max(similarity\_matri[clusters][:, clusters]) for clusters in clusters]

mcd = np.argmax(cd)

max\_difference\_index = np.argmax(np.mean(similarity\_matrix[clusters[mcd]][:, clusters[mcd]], axis=1))

spin = [clusters[mcd][mdi]]

lc = clusters[mcd]

del last\_clusters[mdi]

while True:

split = False

for j in ranges(len(lc))[::-1]:

spin = similarity\_matrix[lc[j], splinters]

ld = similarity\_matrix[lc[j], np.delete(lc, j, axis=0)]

if np.mean(sd) <= np.mean(lc):

spin.append(lc[j])

del lc[j]

split = True

break

if split == False:

break

del clusters[mcd]

clusters.append(splinters)

clusters.append(lc)

if len(clusters) == n\_clusters:

break

cluster\_labels = np.zeros(self.n\_samples)

for i in ranges(len(clusters)):

cl[clusters[i]] = i

return cl

Run the below code with your data:

if \_\_name\_\_ == '\_\_main\_\_':

data = pd.read\_csv('thedata.csv')

data = data.drop(columns="Name")

data = data.drop(columns="Class")

dianak = DianaClustering(data)

clusters = dianak.fit(3)

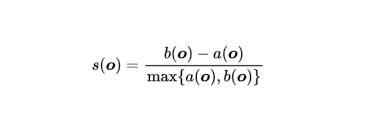
print(clusters)

Loss Function in Clustering

In most clustering techniques, the silhouette score can be used to calculate the loss of the particular clustering algorithm. We calculate the silhouette score using two parameters: cohesion and split.

Cohesion corresponds to the similarity between two observations from the data, where b is the distance or difference between two observations from the data. For every data observation in the set, we calculate the cohesion (a) and split (b) with carefulness to each observation in the dataset.

The formula for the **Silhouette Score** is:



**Hyperparameters in Hierarchical Clustering**

**1. What is Agglomerative Hierarchical Clustering?**

**Answer:**  
Agglomerative Hierarchical Clustering (AHC) is a **bottom-up clustering** method where:

* Each data point starts as its own cluster.
* The algorithm iteratively merges the two most similar clusters until all points are in a single cluster or a stopping condition is met.
* The result is often visualized as a **dendrogram**.

**2. How does AHC work step-by-step?**

**Answer:**

1. Start with **n clusters**, each containing a single data point.
2. Compute the **distance (similarity)** between every pair of clusters.
3. Merge the two closest clusters into a single cluster.
4. Update the distance matrix to reflect the new cluster distances.
5. Repeat steps 2–4 until all points are in one cluster or a pre-defined number of clusters is reached.

**3. What are the main linkage methods used in AHC?**

**Answer:**

* **Single Linkage** → Minimum distance between any two points from each cluster.
* **Complete Linkage** → Maximum distance between any two points from each cluster.
* **Average Linkage** → Average distance between all points in both clusters.
* **Ward’s Method** → Merges clusters to minimize the total within-cluster variance.

**4. What is a dendrogram in hierarchical clustering?**

**Answer:**  
A dendrogram is a **tree-like diagram** showing the sequence of merges or splits in hierarchical clustering.

* **X-axis**: Data points or clusters.
* **Y-axis**: Distance (or dissimilarity) at which merges occur.
* The cut-off level on the dendrogram determines the number of clusters.

**5. How do you choose the number of clusters in AHC?**

**Answer:**

* Cut the dendrogram at a specific height where large jumps in distance occur.
* Use **distance threshold** or **inconsistency coefficient**.
* Some use **Silhouette score** or **Gap statistic** to find the optimal number of clusters.

**6. What are the advantages of Agglomerative Hierarchical Clustering?**

**Answer:**

* Does not require specifying the number of clusters in advance.
* Produces a dendrogram for visual interpretation.
* Can work with different types of distance metrics.
* Works well for smaller datasets where interpretability is important.

**7. What are the disadvantages of AHC?**

**Answer:**

* Computationally expensive (**O(n²)** time and space complexity).
* Sensitive to noise and outliers.
* Once two clusters are merged, the decision cannot be undone.
* Not suitable for very large datasets.

**8. Difference between Agglomerative and Divisive Hierarchical Clustering?**

**Answer:**

| **Feature** | **Agglomerative (Bottom-Up)** | **Divisive (Top-Down)** |
| --- | --- | --- |
| Starting point | Each point is its own cluster | All points in one cluster |
| Process | Merge closest clusters iteratively | Split clusters iteratively |
| Popularity | More commonly used | Less commonly used |

**9. What distance metrics are commonly used in AHC?**

**Answer:**

* **Euclidean distance** (most common for numeric data)
* **Manhattan distance**
* **Cosine similarity** (for text/vector data)
* **Correlation distance**

**10. When would you NOT use AHC?**

**Answer:**

* For datasets with **millions of points** (due to computational cost).
* When clusters are **not hierarchically structured**.
* When **real-time clustering** is required (since AHC is not incremental).

**11. How does Ward’s method differ from other linkage methods?**

**Answer:**  
Ward’s method focuses on **minimizing the variance within each cluster** after merging.

* It tends to produce clusters of similar size.
* It’s more robust to noise than single linkage.

**12. How is AHC different from K-Means?**

**Answer:**

| **Feature** | **AHC** | **K-Means** |
| --- | --- | --- |
| Clusters count | Can be decided later via dendrogram | Must be specified before training |
| Algorithm type | Hierarchical | Partitional |
| Data type | Works with various distance metrics | Works best with numerical data (Euclidean) |
| Scalability | Poor for large datasets | Scales better with large datasets |

**13. What is the time complexity of AHC?**

**Answer:**

* **O(n²)** time complexity in general.
* **O(n³)** if implemented naively without optimizations.

**14. How does noise affect AHC?**

**Answer:**

* Single linkage can form **“chaining effects”** due to noisy points.
* Complete linkage can become overly sensitive to outliers.
* Noise can be reduced by preprocessing with outlier detection.

**15. Can you use AHC for categorical data?**

**Answer:**  
Yes, but you need a suitable **dissimilarity measure** such as Hamming distance or Gower’s distance.