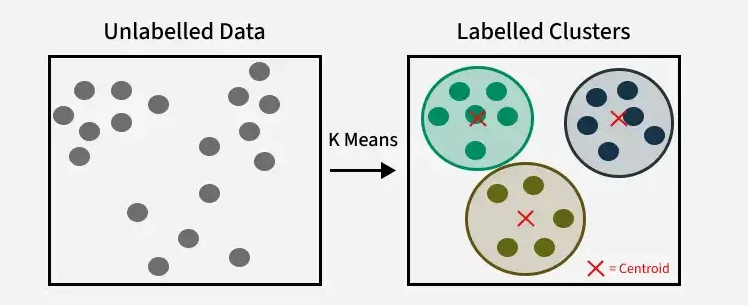
# K-means clustering

[K-Means Clustering](https://www.geeksforgeeks.org/k-means-clustering-introduction/) is an[Unsupervised Machine Learning](https://www.geeksforgeeks.org/ml-types-learning-part-2/) algorithm which groups the unlabeled dataset into different clusters. The article aims to explore the fundamentals and working of k means clustering along with its implementation.

K-means clustering is a technique used to organize data into **groups based on their similarity**. For example **online store uses K-Means to group customers based on purchase frequency and spending creating segments like Budget Shoppers, Frequent Buyers and Big Spenders for personalized marketing.**

The algorithm works by first randomly picking some central points called**centroids**and each data point is then assigned to the closest centroid forming a cluster. After all the points are assigned to a cluster the centroids are updated by finding the average position of the points in each cluster. This process repeats until the centroids stop changing forming clusters. The goal of clustering is to divide the data points into clusters so that similar data points belong to same group.



The algorithm will categorize the items into k groups or clusters of similarity. To calculate that similarity, we will use the [Euclidean distance](https://www.geeksforgeeks.org/euclidean-distance/) as a measurement. The algorithm works as follows:

1. First, we randomly initialize k points, called means or cluster centroids.
2. We categorize each item to its closest mean, and we update the mean’s coordinates, which are the averages of the items categorized in that cluster so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The “points” mentioned above are called means because they are the mean values of the items categorized in them. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set. For example for a feature *x* the items have values in [0,3] we will initialize the means with values for *x* at [0,3].

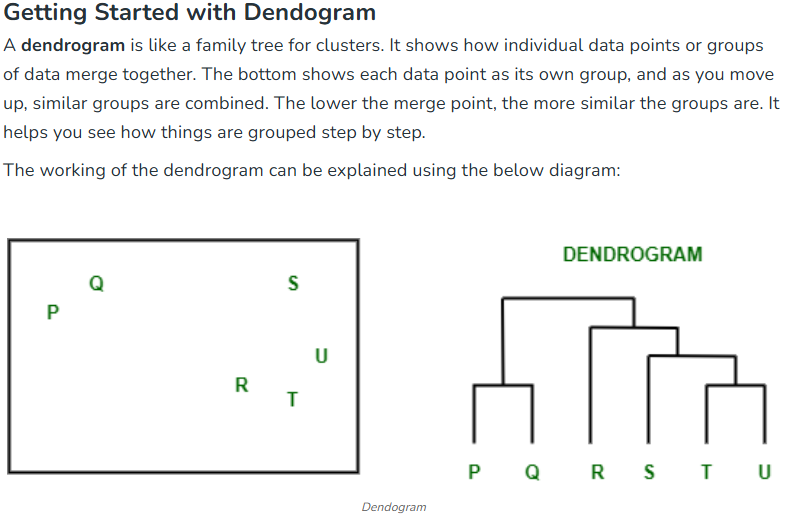
# Hierarchical Clustering

In the real world **data often lacks a target variable making supervised learning impractical.** Have you ever wondered how social networks like Facebook recommend friends or how scientists group similar species together? These are some examples of hierarchical clustering that we will learn about in this article.

**Hierarchical clustering**is a technique used to group similar data points together based on their similarity creating a **hierarchy or tree-like structure**. The key idea is to begin with each data point as its own separate cluster and then progressively merge or split them based on their similarity.

Imagine you have four fruits with different weights: an **apple (100g)**, **a banana (120g), a cherry (50g), and a grape (30g)**. Hierarchical clustering starts by treating each ***fruit as its own group***.

* It then merges the closest groups based on their weights.
* First, the cherry and grape are grouped together because they are the lightest.
* Next, the apple and banana are grouped together.



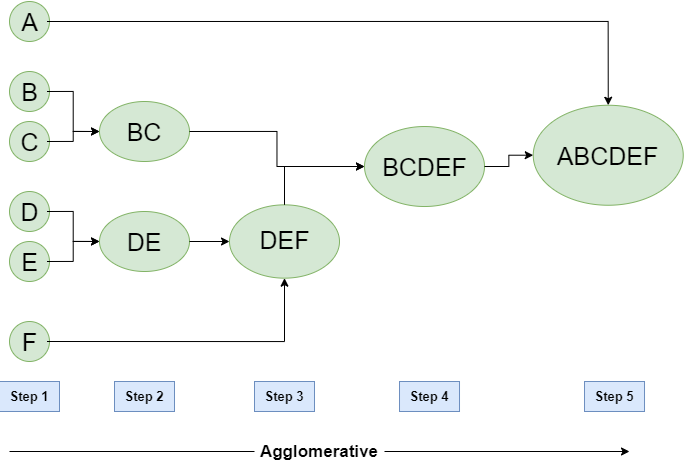
**Types of Hierarchical Clustering**

Now that we understand the basics of hierarchical clustering, let’s explore the two main types of hierarchical clustering.

1. Agglomerative Clustering
2. Divisive clustering

**Hierarchical Agglomerative Clustering**

It is also known as the**bottom-up approach** or **hierarchical agglomerative clustering (HAC)**. Unlike flat clustering hierarchical clustering provides a structured way to group data.



**Workflow for Hierarchical Agglomerative clustering**

1. **Start with individual points**: Each data point is its own cluster. For example if you have 5 data points you start with 5 clusters each containing just one data point.
2. **Calculate distances between clusters**: Calculate the distance between every pair of clusters. Initially since each cluster has one point this is the distance between the two data points.
3. **Merge the closest clusters**: Identify the two clusters with the smallest distance and merge them into a single cluster.
4. **Update distance matrix**: After merging you now have one less cluster. Recalculate the distances between the new cluster and the remaining clusters.
5. **Repeat steps 3 and 4**: Keep merging the closest clusters and updating the distance matrix until you have only one cluster left.
6. **Create a dendrogram**: As the process continues you can visualize the merging of clusters using a tree-like diagram called a **dendrogram**. It shows the hierarchy of how clusters are merged.

**Hierarchical Divisive clustering**

It is also known as a **top-down approach**. This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been split into singleton clusters.

**Workflow for Hierarchical Divisive clustering :**

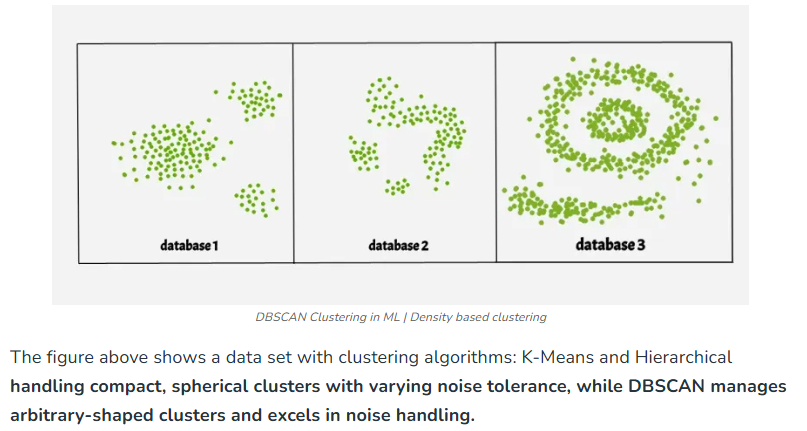
1. **Start with all data points in one cluster**: Treat the entire dataset as a single large cluster.
2. **Split the cluster**: Divide the cluster into two smaller clusters. The division is typically done by finding the two most dissimilar points in the cluster and using them to separate the data into two parts.
3. **Repeat the process**: For each of the new clusters, repeat the splitting process:
   1. Choose the cluster with the most dissimilar points.
   2. Split it again into two smaller clusters.
4. **Stop when each data point is in its own cluster**: Continue this process until every data point is its own cluster, or the stopping condition (such as a predefined number of clusters) is met.

# DBSCAN

DBSCAN is a **density-based clustering algorithm** that**groups data points that are closely packed together and marks outliers as noise** based on their density in the feature space. It  identifies clusters as dense regions in the data space, separated by areas of lower density.

Unlike K-Means or hierarchical clustering, which assume clusters are **compact and spherical**, DBSCAN excels in handling real-world data irregularities such as:

* **Arbitrary-Shaped Clusters**: Clusters can take any shape, not just circular or convex.
* **Noise and Outliers**: It effectively identifies and handles noise points without assigning them to any cluster.



**Key Parameters in DBSCAN**

* **1. eps**: This defines the radius of the neighborhood around a data point.

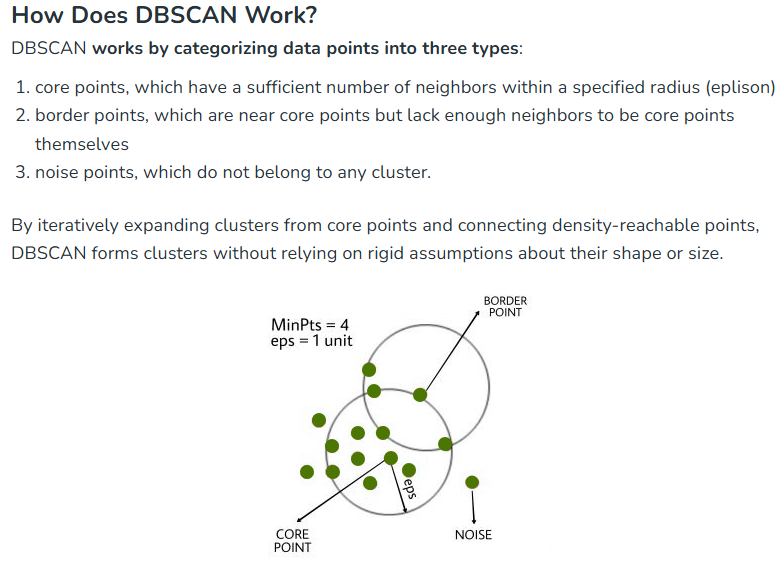
If the distance between two points is less than or equal to **eps**, they are considered neighbors. Choosing the right **eps** is crucial:

* If **eps** is too small, most points will be classified as noise.
* If **eps** is too large, clusters may merge, and the algorithm may fail to distinguish between them.

A common method to determine **eps** is by analyzing the **k-distance graph**.

* **2. MinPts**: This is the minimum number of points required within the **eps** radius to form a dense region.

A general rule of thumb is to set MinPts >= D+1, where **D** is the number of dimensions in the dataset. For most cases, a minimum value of **MinPts = 3** is recommended.



**Steps in the DBSCAN Algorithm**

1. **Identify Core Points**: For each point in the dataset, count the number of points within its **eps** neighborhood. If the count meets or exceeds **MinPts**, mark the point as a **core point**.
2. **Form Clusters**: For each core point that is not already assigned to a cluster, create a new cluster. Recursively find all **density-connected points** (points within the **eps** radius of the core point) and add them to the cluster.
3. **Density Connectivity**: Two points, **a** and **b**, are **density-connected** if there exists a chain of points where each point is within the **eps** radius of the next, and at least one point in the chain is a core point. This chaining process ensures that all points in a cluster are connected through a series of dense regions.
4. **Label Noise Points**: After processing all points, any point that does not belong to a cluster is labeled as **noise**.

# PCA

Having too many features in data can cause problems like overfitting (good on training data but poor on new data), slower computation, and lower accuracy. This is called the [curse of dimensionality](https://www.geeksforgeeks.org/videos/curse-of-dimensionality-in-machine-learning/), where more features exponentially increase the data needed for reliable results.

To tackle this problem, we use Feature engineering Techniques ,such as feature selection (choosing the most important features) and feature extraction (creating new features from the original ones). One popular feature extraction method is dimensionality reduction, which reduces the number of features while keeping as much important information as possible.

One of the most widely used dimensionality reduction techniques is **Principal Component Analysis (PCA)**.

**How PCA Works for Dimensionality Reduction?**

PCA is a statistical technique introduced by mathematician Karl Pearson in 1901. ***It works by transforming high-dimensional data into a lower-dimensional space while maximizing the variance (or spread) of the data in the new space***. This helps preserve the most important patterns and relationships in the data.

1. **Standardize the Data**:
   * Make sure all features are on the same scale (e.g., using StandardScaler).
   * This is important because PCA depends on the variance of the data.
2. **Find Principal Components**:
   * PCA rotates the data onto new axes (principal components).
   * The **1st principal component** captures the most variability (spread) in the data.
   * The **2nd principal component** captures the next highest variability, and so on.
   * These components are perpendicular to each other (orthogonal).
3. **Rank the Components**:
   * PCA ranks the components based on the variance they capture.
   * You decide how many components to keep (e.g., keep 2 components for visualization or enough to explain 95% of the variance).
4. **Transform the Data**:
   * The data is projected onto the selected principal components (lower dimensions).
   * You get a simplified version of the data.

**Key Terms:**

1. **Variance**: How much the data varies. PCA tries to maximize variance in the new components.
2. **Eigenvalues**: Measure the importance (variance explained) of each principal component.
3. **Eigenvectors**: Directions of the new axes (principal components).
4. **Explained Variance Ratio**: The percentage of total variance explained by each principal component.

**Example:**

Imagine you have a dataset with two features:

* **Feature 1**: Height
* **Feature 2**: Weight

PCA might combine these into:

* **Principal Component 1 (PC1)**: A combination of height and weight, explaining most of the variability (e.g., overall "size").
* **Principal Component 2 (PC2)**: A secondary trend (e.g., differences in body proportions).