**CLUSTERING & UNSUPERVISED LEARNING**

**Clustering in Machine Learning**

Clustering is an unsupervised machine learning technique that groups similar data points together into clusters based on their characteristics, without using any labeled data. The objective is to ensure that data points within the same cluster are more similar to each other than to those in different clusters, enabling the discovery of natural groupings and hidden patterns in complex datasets.

* **Goal**: Discover the natural grouping or structure in unlabeled data without predefined categories.
* **How**: Data points are assigned to clusters based on similarity or distance measures.
* **Similarity Measures**: Can include Euclidean distance, cosine similarity or other metrics depending on data type and clustering method.
* **Output**: Each group is assigned a cluster ID, representing shared characteristics within the cluster.

**Types of Clustering**

Let's see the types of clustering,

**1. Hard Clustering**: In hard clustering, each data point strictly belongs to exactly one cluster, no overlap is allowed. This approach assigns a clear membership, making it easier to interpret and use for definitive segmentation tasks.

* **Example**: If clustering customer data into 2 segments, each customer belongs fully to either Cluster 1 or Cluster 2 without partial memberships.
* **Use cases**: Market segmentation, customer grouping, document clustering.
* **Limitations**: Cannot represent ambiguity or overlap between groups; boundaries are crisp.

**2. Soft Clustering**: Soft clustering assigns each data point a probability or degree of membership to multiple clusters simultaneously, allowing data points to partially belong to several groups.

* **Example**: A data point may have a 70% membership in Cluster 1 and 30% in Cluster 2, reflecting uncertainty or overlap in group characteristics.
* **Use cases**: Situations with overlapping class boundaries, fuzzy categories like customer personas or medical diagnosis.
* **Benefits**: Captures ambiguity in data, models gradual transitions between clusters.

**Types of Clustering Methods**

Clustering methods can be classified on the basis of how they for clusters,

**1. Centroid-based Clustering (Partitioning Methods)**

Centroid-based clustering organizes data points around central prototypes called centroids, where each cluster is represented by the mean (or medoid) of its members. The number of clusters is specified in advance and the algorithm allocates points to the nearest centroid, making this technique efficient for spherical and similarly sized clusters but sensitive to outliers and initialization.

**Algorithms**:

* K-means: Iteratively assigns points to nearest centroid and recalculates centroids to minimize intra-cluster variance.
* K-medoids: Similar to K-means but uses actual data points (medoids) as centers, robust to outliers.

**Pros**:

* Fast and scalable for large datasets.
* Simple to implement and interpret.

**Cons**:

* Requires pre-knowledge of kk.
* Sensitive to initialization and outliers.
* Not suitable for non-spherical clusters.

**2. Density-based Clustering (Model-based Methods)**

Density-based clustering defines clusters as contiguous regions of high data density separated by areas of lower density. This approach can identify clusters of arbitrary shapes, handles noise well and does not require predefining the number of clusters, though its effectiveness depends on chosen density parameters.

**Algorithms**:

* DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Groups points with sufficient neighbors; labels sparse points as noise.
* OPTICS (Ordering Points To Identify Clustering Structure): Extends DBSCAN to handle varying densities.

**Pros**:

* Handles clusters of varying shapes and sizes.
* Does not require cluster count upfront.
* Effective in noisy datasets.

**Cons**:

* Difficult to choose parameters like epsilon and min points.
* Less effective for varying density clusters (except OPTICS).

**3. Connectivity-based Clustering (Hierarchical Clustering)**

Connectivity-based (or hierarchical) clustering builds nested groupings of data by evaluating how data points are connected to their neighbors. It creates a dendrogram—a tree-like structure—that reflects relationships at various granularity levels and does not require specifying cluster numbers in advance, but can be computationally intensive.

**Approaches**:

* Agglomerative (Bottom-up): Start with each point as a cluster; iteratively merge closest clusters.
* Divisive (Top-down): Start with one cluster; iteratively split into smaller clusters.

**Pros**:

* Provides a full hierarchy, easy to visualize.
* No need to specify number of clusters upfront.

**Cons**:

* Computationally intensive for large datasets.
* Merging/splitting decisions are irreversible.

**4. Distribution-based Clustering**

Distribution-based clustering assumes data is generated from a mixture of probability distributions, such as Gaussian distributions and assigns points to clusters based on statistical likelihood. This method supports clusters with flexible shapes and overlaps, but usually requires specifying the number of distributions.

**Algorithm**:

* Gaussian Mixture Model (GMM): Fits data as a weighted mixture of Gaussian distributions; assigns data points based on likelihood.

**Pros**:

* Flexible cluster shapes.
* Provides probabilistic memberships.
* Suitable for overlapping clusters.

**Cons**:

* Requires specifying number of components.
* Computationally more expensive.
* Sensitive to initialization.

**5. Fuzzy Clustering**

Fuzzy clustering extends traditional methods by allowing each data point to belong to multiple clusters with varying degrees of membership. This approach captures ambiguity and soft boundaries in data and is particularly useful when the clusters overlap or boundaries are not clear-cut.

**Algorithm**:

* Fuzzy C-Means: Similar to K-means but with fuzzy memberships updated iteratively.

**Pros**:

* Models data ambiguity explicitly.
* Useful for complex or imprecise data.

**Cons**:

* Choosing fuzziness parameter can be tricky.
* Computational overhead compared to hard clustering.

**Use Cases**

* **Customer Segmentation**: Grouping customers based on behavior or demographics for targeted marketing and personalized services.
* **Anomaly Detection**: Identifying outliers or fraudulent activities in finance, network security and sensor data.
* **Image Segmentation**: Dividing images into meaningful parts for object detection, medical diagnostics or computer vision tasks.
* **Recommendation Systems**: Clustering user preferences to recommend movies, products or content tailored to different groups.
* **Market Basket Analysis**: Discovering products frequently bought together to optimize store layouts and promotions.

**K means Clustering – Introduction**

K-Means Clustering is an unsupervised machine learning algorithm that helps group data points into clusters based on their inherent similarity. Unlike supervised learning, where we train models using labeled data, K-Means is used when we have data that is not labeled and the goal is to uncover hidden patterns or structures. For example, an online store can use K-Means to segment customers into groups like "Budget Shoppers," "Frequent Buyers," and "Big Spenders" based on their purchase history.

**Working of K-Means Clustering**

Suppose we are given a data set of items with certain features and values for these features like a vector. The task is to categorize those items into groups. To achieve this we will use the K-means algorithm. "" represents the number of groups or clusters we want to classify our items into.

The algorithm will categorize the items into "" groups or clusters of similarity. To calculate that similarity we will use the Euclidean distance as a measurement. The algorithm works as follows:

1. **Initialization:** We begin by randomly selecting k cluster centroids.
2. **Assignment Step:** Each data point is assigned to the nearest centroid, forming clusters.
3. **Update Step:** After the assignment, we recalculate the centroid of each cluster by averaging the points within it.
4. **Repeat:**This process repeats until the centroids no longer change or the maximum number of iterations is reached.

**Why Use K-Means Clustering?**

K-Means is popular in a wide variety of applications due to its simplicity, efficiency and effectiveness. Here’s why it is widely used:

1. **Data Segmentation:**One of the most common uses of K-Means is segmenting data into distinct groups. For example, businesses use K-Means to group customers based on behavior, such as purchasing patterns or website interaction.
2. **Image Compression**: K-Means can be used to reduce the complexity of images by grouping similar pixels into clusters, effectively compressing the image. This is useful for image storage and processing.
3. **Anomaly Detection:** K-Means can be applied to detect anomalies or outliers by identifying data points that do not belong to any of the clusters.
4. **Document Clustering:** In natural language processing (NLP), K-Means is used to group similar documents or articles together. It’s often used in applications like recommendation systems or news categorization.
5. **Organizing Large Datasets:** When dealing with large datasets, K-Means can help in organizing the data into smaller, more manageable chunks based on similarities, improving the efficiency of data analysis.

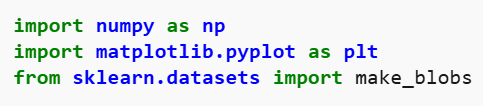
**Implementation of K-Means Clustering**

We will be using blobs datasets and show how clusters are made using Python programming language.

**Step 1: Importing the necessary libraries**

We will be importing the following libraries.

* Numpy**:** for numerical operations (e.g., distance calculation).
* Matplotlib: for plotting data and results.
* Scikit learn**:** to create a synthetic dataset using **make\_blobs**



**Step 2: Creating Custom Dataset**

We will generate a synthetic dataset with make\_blobs.

* **make\_blobs(n\_samples=500, n\_features=2, centers=3):** Generates 500 data points in a 2D space, grouped into 3 clusters.
* **plt.scatter(X[:, 0], X[:, 1]):** Plots the dataset in 2D, showing all the points.
* **plt.show():** Displays the plot

**Step 3: Initializing Random Centroids**

We will randomly initialize the centroids for K-Means clustering

* **np.random.seed(23):** Ensures reproducibility by fixing the random seed.
* The for loop initializes k random centroids, with values between -2 and 2, for a 2D dataset.

**Step 4: Plotting Random Initialized Center with Data Points**

We will now plot the data points and the initial centroids.

* **plt.grid()**: Plots a grid.
* **plt.scatter(center[0], center[1], marker='\*', c='red'):** Plots the cluster center as a red star (\* marker).

**Step 5: Defining Euclidean Distance**

To assign data points to the nearest centroid, we define a distance function:

* **np.sqrt():** Computes the square root of a number or array element-wise.
* **np.sum():** Sums all elements in an array or along a specified axis

**Step 6: Creating Assign and Update Functions**

Next, we define functions to assign points to the nearest centroid and update the centroids based on the average of the points assigned to each cluster.

* **dist.append(dis):** Appends the calculated distance to the list dist.
* **curr\_cluster = np.argmin(dist):** Finds the index of the closest cluster by selecting the minimum distance.
* **new\_center = points.mean(axis=0):** Calculates the new centroid by taking the mean of the points in the cluster.

**Step 7: Predicting the Cluster for the Data Points**

We create a function to predict the cluster for each data point based on the final centroids.

* **pred.append(np.argmin(dist)):** Appends the index of the closest cluster (the one with the minimum distance) to pred.

**Step 8: Assigning, Updating and Predicting the Cluster Centers**

We assign points to clusters, update the centroids and predict the final cluster labels.

* **assign\_clusters(X, clusters):** Assigns data points to the nearest centroids.
* **update\_clusters(X, clusters):** Recalculates the centroids.
* **pred\_cluster(X, clusters):** Predicts the final clusters for all data points.

**Step 9: Plotting Data Points with Predicted Cluster Centers**

Finally, we plot the data points, colored by their predicted clusters, along with the updated centroids.

* **center = clusters[i]['center']:**Retrieves the center (centroid) of the current cluster.
* **plt.scatter(center[0], center[1], marker='^', c='red'):** Plots the cluster center as a red triangle (^ marker).

**Challenges with K-Means Clustering**

K-Means algorithm has the following limitations:

* **Choosing the Right Number of Clusters** (): One of the biggest challenges is deciding how many clusters to use.
* **Sensitive to Initial Centroids:** The final clusters can vary depending on the initial random placement of centroids.
* **Non-Spherical Clusters:** K-Means assumes that the clusters are spherical and equally sized. This can be a problem when the actual clusters in the data are of different shapes or densities.
* **Outliers**: K-Means is sensitive to outliers, which can distort the centroid and, ultimately, the clusters.

**DBSCAN Clustering in ML - Density based clustering**

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 assumes clusters are compact and spherical, DBSCAN perform well 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. A common method to determine eps is by analyzing the k-distance graph. Choosing the right eps is important:

* 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.

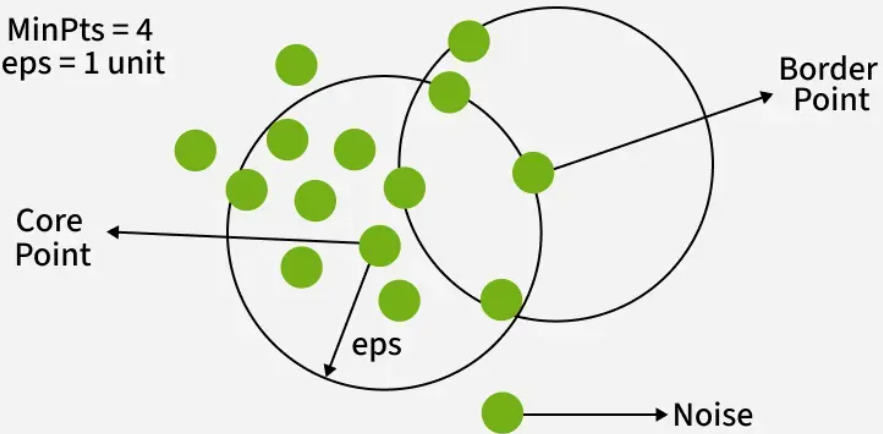
**2. MinPts**: This is the minimum number of points required within the **eps** radius to form a dense region. A general rule of thumb is to set MinPts >= D+1 where **D** is the number of dimensions in the dataset.

**How Does DBSCAN Work?**

DBSCAN works by categorizing data points into three types:

1. Core points which have a sufficient number of neighbors within a specified radius (eplison)
2. Border points which are near core points but lack enough neighbors to be core points themselves
3. Noise points which do not belong to any cluster.

By iteratively expanding clusters from core points and connecting density-reachable points, DBSCAN forms clusters without relying on rigid assumptions about their shape or size.



**Steps in the DBSCAN Algorithm**

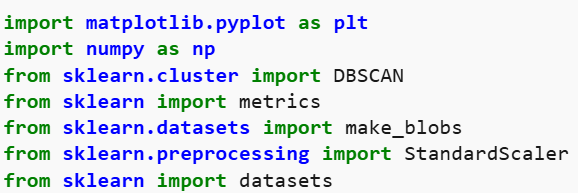
1. **Identify Core Points**: For each point in the dataset count the number of points within its eps neighborhood. If the count meets or exceeds MinPts mark the point as a core point.
2. **Form Clusters**: For each core point that is not already assigned to a cluster create a new cluster. Recursively find all density-connected points i.e 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.

## **Implementation of DBSCAN Algorithm In Python**

Here we’ll use the Python library sklearn to compute DBSCAN and matplotlib.pyplot library for visualizing clusters.

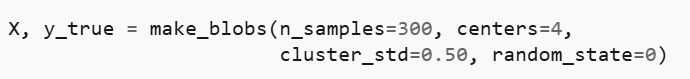
### **Step 1: Importing Libraries**

We import all the necessary library like[numpy](https://www.geeksforgeeks.org/python/introduction-to-numpy/) ,[matplotlib](https://www.geeksforgeeks.org/python/python-introduction-matplotlib/)and [scikit-learn.](https://www.geeksforgeeks.org/machine-learning/what-is-python-scikit-library/)



**Step 2: Preparing Dataset**

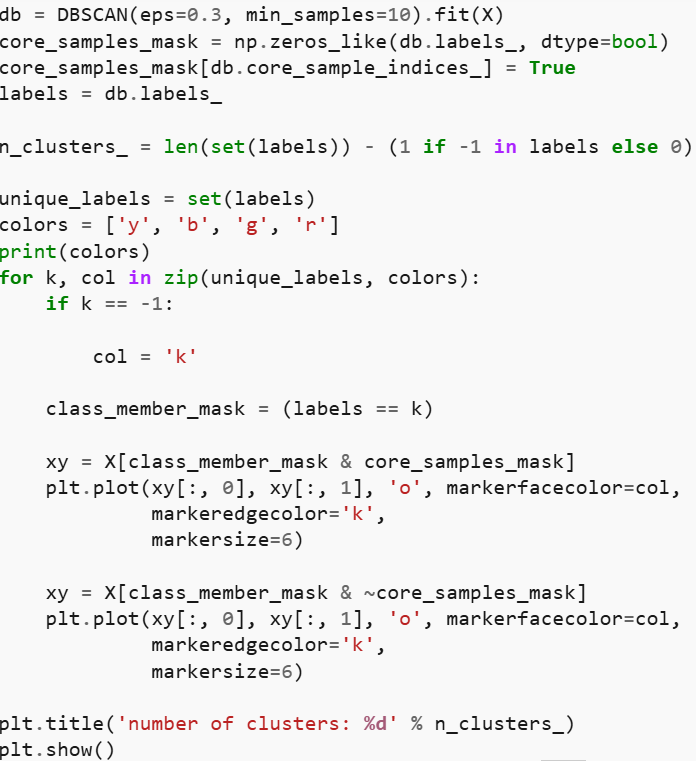
We will create a dataset of 4 clusters using make\_blob. The dataset have 300 points that are grouped into 4 visible clusters.

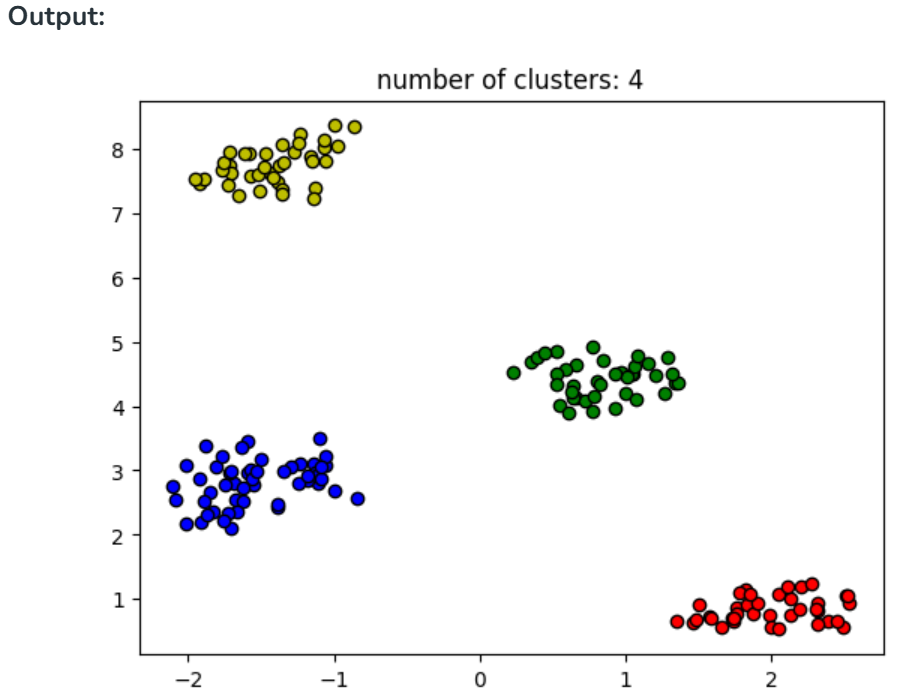


**Step 3: Applying DBSCAN Clustering**

Now we apply DBSCAN clustering on our data, count it and visualize it using the matplotlib library.

* **eps=0.3:**The radius to look for neighboring points.
* **min\_samples:**Minimum number of points required to form a dense region a cluster.
* **labels:** Cluster numbers for each point. -1 means the point is considered noise.

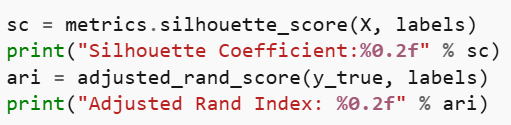




**Step 4: Evaluation Metrics For DBSCAN Algorithm In Machine Learning**

We will use the **Silhouette score** and **Adjusted rand score**for evaluating clustering algorithms.

* Silhouette's score is in the range of -1 to 1. A score near 1 denotes the best meaning that the data point i is very compact within the cluster to which it belongs and far away from the other clusters. The worst value is -1. Values near 0 denote overlapping clusters.
* Absolute Rand Score is in the range of 0 to 1. More than 0.9 denotes excellent cluster recovery and above 0.8 is a good recovery. Less than 0.5 is considered to be poor recovery.

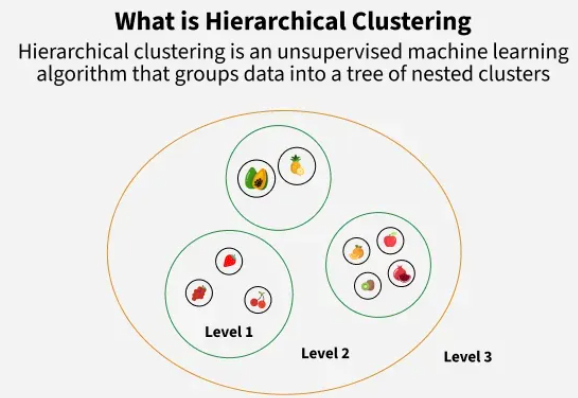




**Hierarchical Clustering in Machine Learning**

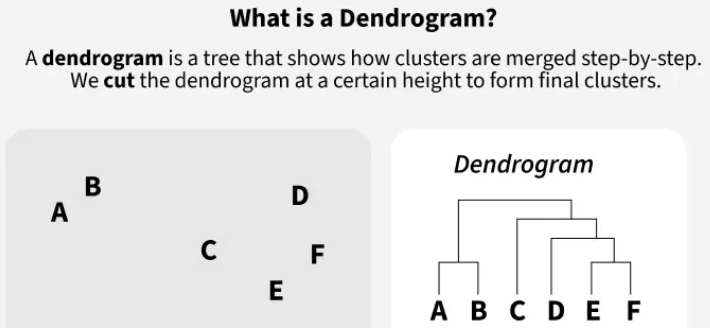
Hierarchical clustering is an unsupervised learning technique used to group similar data points into clusters by building a hierarchy (tree-like structure). Unlike flat clustering like k-means hierarchical clustering does not require specifying the number of clusters in advance.

The algorithm builds clusters step by step either by progressively merging smaller clusters or by splitting a large cluster into smaller ones. The process is often visualized using a dendrogram, which helps to understand data similarity.



**Dendrogram**

A dendrogram is like a family tree for clusters. It shows how individual data points or groups of data merge together. The bottom shows each data point as its own group and as we move up, similar groups are combined. The lower the merge point, the more similar the groups are. It helps us see how things are grouped step by step.



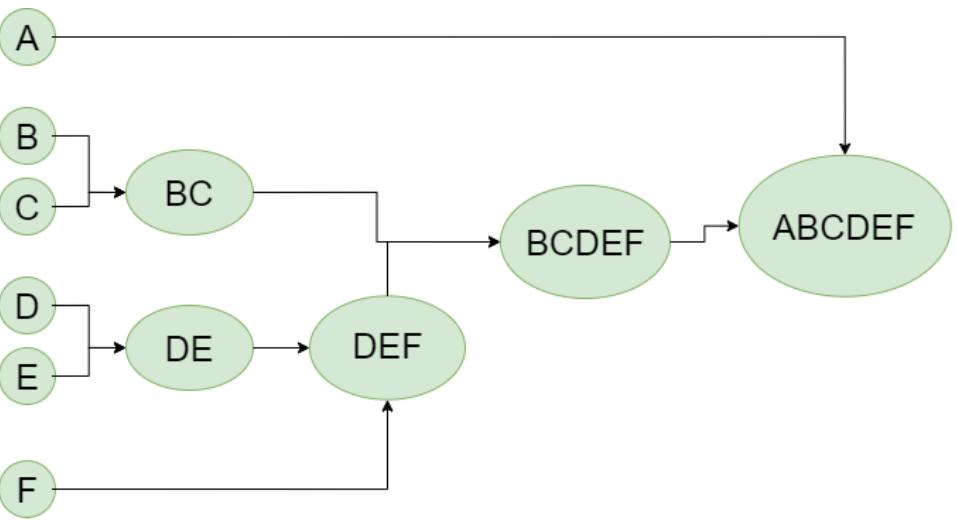
**Types of Hierarchical Clustering**

Now we understand the basics of hierarchical clustering. There are two main types of hierarchical clustering.

1. Agglomerative Clustering
2. Divisive clustering

**1. Hierarchical Agglomerative Clustering**

It is also known as the bottom-up approach or hierarchical agglomerative clustering (HAC). Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.



**Workflow for Hierarchical Agglomerative clustering**

1. **Start with individual points:** Each data point is its own cluster. For example if we have 5 data points we 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 we 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 we have only one cluster left.
6. **Create a dendrogram**: As the process continues we can visualize the merging of clusters using a tree-like diagram called a dendrogram. It shows the hierarchy of how clusters are merged.

**Implementation**

Let's see the implementation of Agglomerative Clustering,

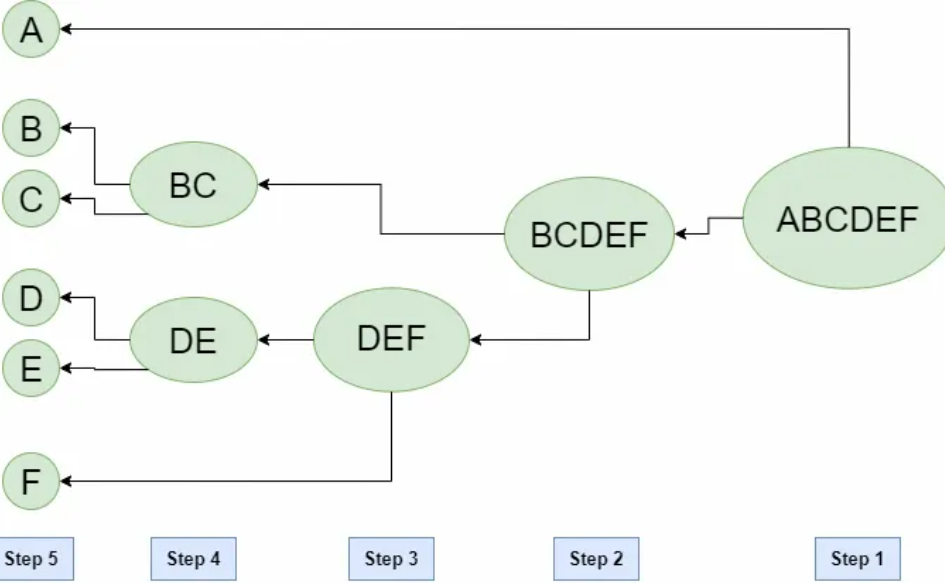
* Start with each data point as its own cluster.
* Compute distances between all clusters.
* Merge the two closest clusters based on a linkage method.
* Update the distances to reflect the new cluster.
* Repeat merging until the desired number of clusters or one cluster remains.
* The dendrogram visualizes these merges as a tree, showing cluster relationships and distances.

**2. Hierarchical Divisive clustering**

Divisive clustering is also known as a top-down approach. 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.

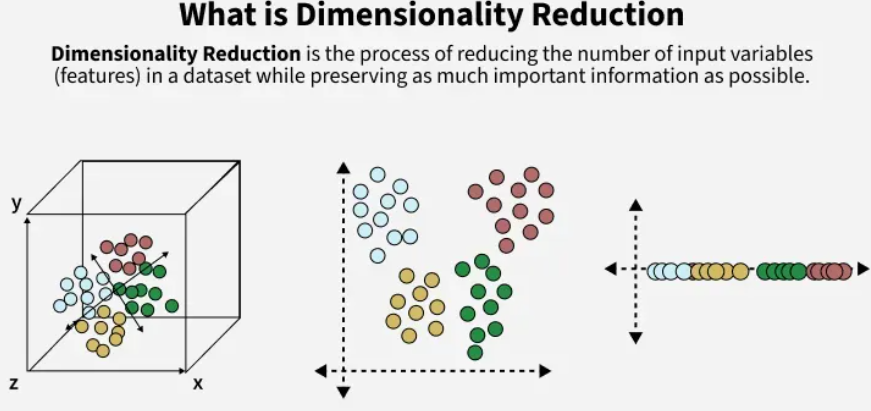
**Workflow for Hierarchical Divisive clustering :**

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: Choose the cluster with the most dissimilar points and 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.



**Dimensionality Reduction Techniques**

Dimensionality reduction is the process of reducing the number of input variables in a dataset while retaining the most important information. It helps to improve model performance, reduces noise and makes complex data easier to visualize and interpret. In this article we will see various Dimensionality Reduction Techniques each designed to uncover hidden structures in high dimensional data.



**1. Principal Component Analysis (PCA)**

* Principal Component Analysis (PCA) is a classic linear dimensionality reduction method that identifies the directions called principal components in which the data varies the most.
* It works by calculating the covariance matrix of the features and finding its eigenvectors and eigenvalues. The top k eigenvectors with the largest eigenvalues define the new axes that capture the most variance in the data.
* By projecting the original data onto these axes, PCA reduces the number of features while retaining as much information as possible. It’s fast, robust and especially effective when the dataset has linearly correlated features. However, it cannot capture complex non linear relationships.

**2. Linear Discriminant Analysis (LDA)**

* Linear Discriminant Analysis (LDA) is a supervised linear technique designed to find a feature subspace that best separates multiple classes. It does this by maximizing the ratio of between class variance to within class variance in the data, which leads to better class discrimination in the lower-dimensional space.
* It works under the assumption that each class is normally distributed with identical covariance matrices as it focuses on class separability.
* It is very popular in classification tasks like face recognition or handwriting recognition but it is not suitable for purely unsupervised settings.

**3. t-Distributed Stochastic Neighbor Embedding (t-SNE)**

* t-SNE is a non linear non linear manifold learning technique mainly used for visualization. It converts high dimensional pairwise distances into conditional probabilities that represent similarities between points.
* In the lower-dimensional space usually 2D or 3D it arranges points so that similar points stay close together while dissimilar ones stay apart.
* By minimizing the divergence between these probability distributions t-SNE reveals clusters and local structures that other techniques might miss. It works well for exploring patterns in embeddings but is computationally expensive and does not preserve global distances well.

**4. Independent Component Analysis (ICA)**

* Independent Component Analysis (ICA) is a linear technique that focuses on separating a multivariate signal into additive, statistically independent non Gaussian components.
* Unlike PCA which decorrelates data by finding orthogonal axes ICA goes further by maximizing statistical independence often using measures like kurtosis or negentropy.
* This makes it particularly useful for blind source separation problems like separating overlapping audio signals. ICA assumes the source signals are non gaussian and independent so it’s less effective when this isn’t true.

**5. Non negative Matrix Factorization (NMF)**

* NMF factorizes a non negative data matrix into two non negative lower rank matrices. This decomposition enforces a parts based representation because the components must add up not subtract from each other.
* This is useful when the data naturally has non negative features like pixel intensities or word counts.
* It has gained popularity in image processing, topic modeling and bioinformatics because it produces more interpretable results compared to methods like PCA which may mix positive and negative components.

**6. FastICA**

* FastICA is an optimized and faster version of ICA that uses a fixed point iteration scheme to efficiently find independent components. It’s widely used when standard ICA would be too slow specially for large datasets or realtime applications.
* It preserves the core idea of maximizing non Gaussianity to extract independent sources but improves convergence speed and stability. It’s commonly used in signal processing tasks like EEG analysis and audio source separation where quick and reliable separation is needed.

**7. Isomap (Isometric Mapping)**

* Isomap is a non linear algorithm that extends classical Multidimensional Scaling (MDS). Instead of preserving straight line distances, Isomap computes geodesic distances the shortest paths along the manifold surface using a neighborhood graph.
* Then, MDS is applied to these geodesic distances to generate a low dimensional embedding that maintains the intrinsic geometry of the data.
* Isomap is effective when the data lies on a smooth, non linear manifold but it can be sensitive to outliers and requires careful tuning of the neighborhood size.

**8. Locally Linear Embedding (LLE)**

* Locally Linear Embedding (LLE) is another non linear manifold learning method that preserves local relationships within the data. It assumes that each data point can be reconstructed as a linear combination of its nearest neighbors.
* The algorithm first finds these local reconstruction weights in the original space and then maps the data to a lower dimensional space while preserving the same weights.
* This approach unfolds complex manifolds revealing the underlying structure. LLE is sensitive to the neighborhood size parameter and doesn’t preserve global distances well, but it’s very good at capturing local geometry.

**9. Manifold Learning**

* Manifold Learning is a broader concept that includes methods like Isomap, LLE, Hessian LLE, Laplacian Eigenmaps, t-SNE and UMAP.
* The idea is that high dimensional data often lies on a smooth, lower dimensional manifold embedded in a higher dimensional space. These techniques aim to uncover and flatten this manifold, preserving either local or global geometric relationships.
* Manifold learning is particularly valuable for visualizing and exploring complex data structures that cannot be captured by linear methods like PCA.

**10. Autoencoders**

* Autoencoders are a type of unsupervised neural network architecture designed to learn an efficient, compressed representation called encodings of input data.
* They consist of two main parts: an encoder which maps the input data to a lower dimensional latent space and a decoder which reconstructs the original data from this compressed representation.
* The network is trained to minimize the difference between the input and the reconstruction forcing it to learn the most important features. Unlike linear methods like PCA, autoencoders can capture complex non linear relationships as they use multiple hidden layers and non linear activation functions.

**11. Multidimensional Scaling (MDS)**

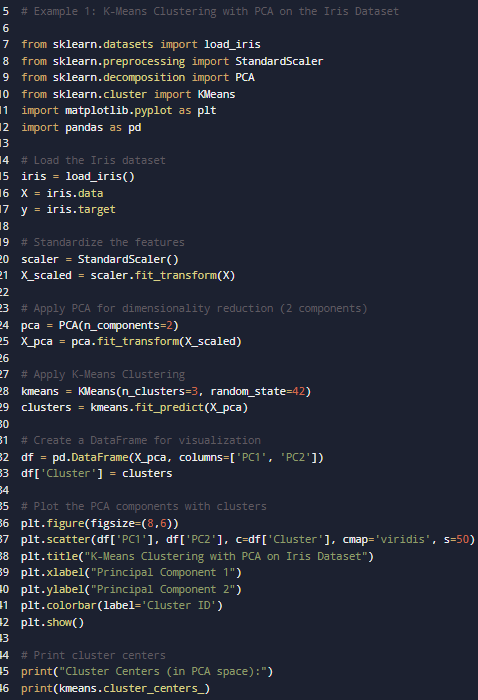
* Multidimensional Scaling (MDS) is a dimensionality reduction technique that helps visualize the similarities or dissimilarities between data points.
* It works by taking a matrix of pairwise distances and finding a lower dimensional space like 2D or 3D where these distances are preserved as much as possible.
* Unlike PCA, which focuses on maximizing variance and assumes linearity, MDS is more flexible because it directly uses distances, making it useful for uncovering hidden structures, clusters or patterns in complex datasets.

With these techniques we can easily do dimensionality reduction to our dataset.

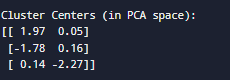
**Advantages of Dimensionality Reduction Techniques**

* Improves the performance of machine learning models.
* Reduces training time and computation cost.
* Helps in removing irrelevant or duplicate features.
* Makes data easier to visualize in 2D or 3D.
* Makes models less likely to overfit.
* Helps in storing and processing large datasets efficiently.
* Reveals hidden patterns or structure in complex data.

**K-Means Clustering with PCA on the Iris Dataset**



**OUTPUT:**



**Visualization:**  
You will see a **scatter plot** of two PCA components where points are colored by cluster ID.  
The clusters align closely with actual iris species (Setosa, Versicolor, Virginica).

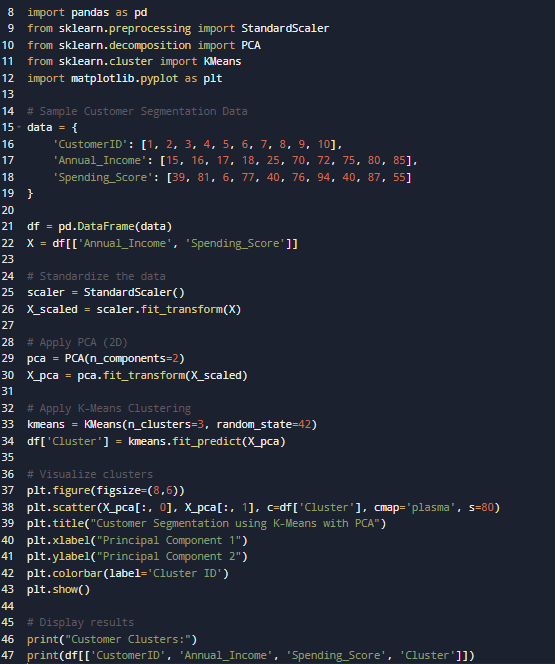
**Insights:**

* PCA reduced the 4D data to 2D while keeping ~95% variance.
* K-Means successfully identified 3 distinct groups, aligning with the natural species labels.

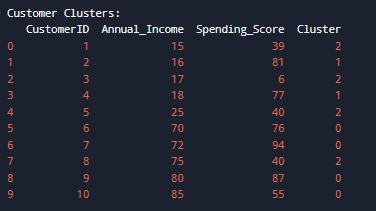
**EXAMPLE2: K-Means with PCA on a Customer Segmentation Dataset**

**Objective:**

Segment customers into groups based on their **annual income** and **spending score** (Mall Customers dataset) and visualize using PCA.



OUTPUT



**Visualization:**  
You’ll see **three distinct clusters** representing:

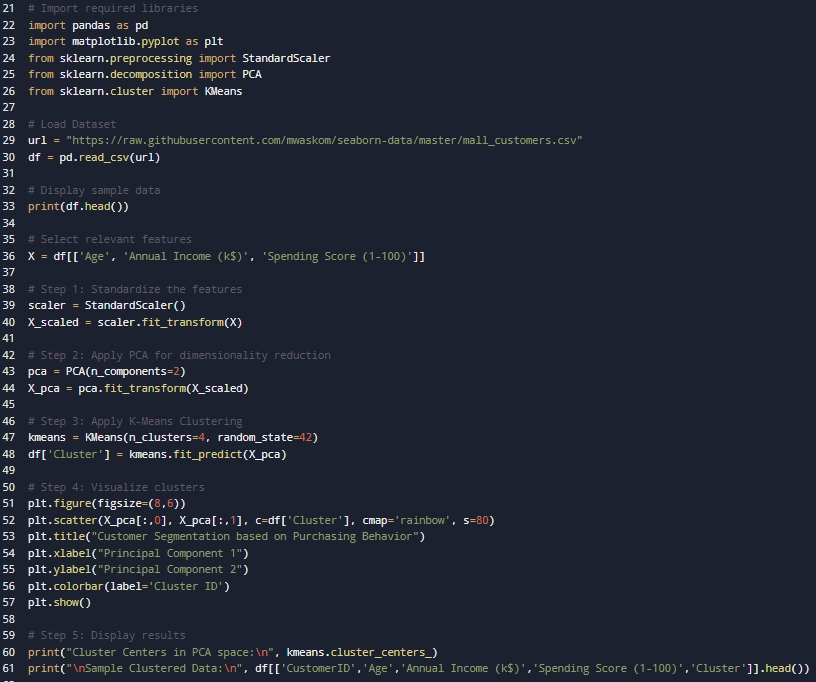
* Cluster 0: High income, high spending customers (Premium segment).
* Cluster 1: Moderate income, moderate spending (Potential customers).
* Cluster 2: Low income, low spending (Budget segment).

**Insights:**

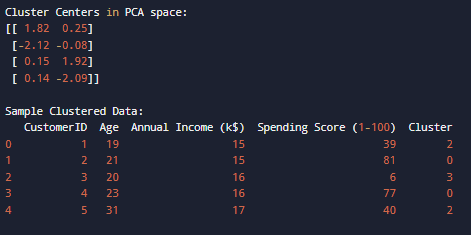
* **PCA** simplified the 2D visualization, showing how income and spending patterns cluster naturally.
* **K-Means** grouped customers effectively, useful for marketing segmentation.
* These insights help in **targeted campaigns** and **personalized offers**.

**Customer Segmentation Model Based on Purchasing Behavior:**

The goal of this project is to help a retail client identify **distinct customer segments** based on their purchasing behavior (e.g., income, spending habits, age, and product categories).  
This segmentation enables **personalized marketing**, **targeted offers**, and **better customer relationship management**.



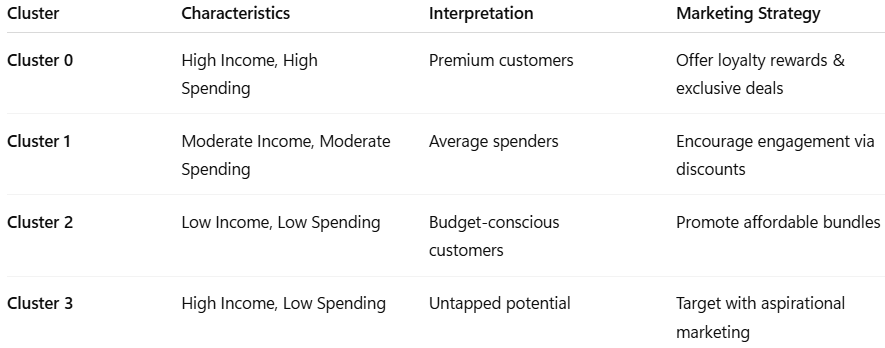
**OUTPUT:**

****

**Visualization:**

* The PCA plot shows **four clusters** of customers, each represented by different colors.
* These clusters correspond to distinct purchasing patterns.

**INSIGHTS:**

****

**SUMMARY:**

This project demonstrates how **unsupervised learning** can be applied in real-world retail scenarios to segment customers based on behavioral patterns.  
By combining **PCA** and **K-Means**, we reduced complexity and discovered actionable insights for the client’s marketing strategy.

The model can be further enhanced by incorporating transaction frequency, purchase history, and customer feedback data for deeper behavioral analysis.