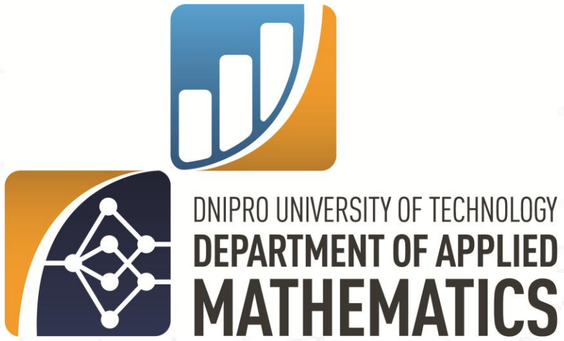


CLUSTERING:
K-means, Hierarchical Clustering

Course "Machine Learning: From Mathematical Foundations to Implementation in PYTHON"

Lecture by prof. Dmytro Babets

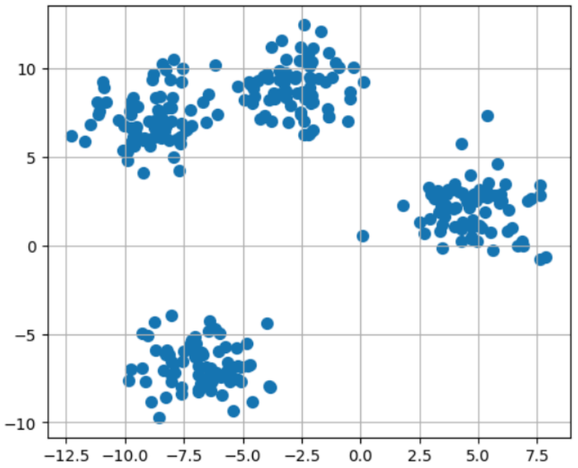
1. Clustering

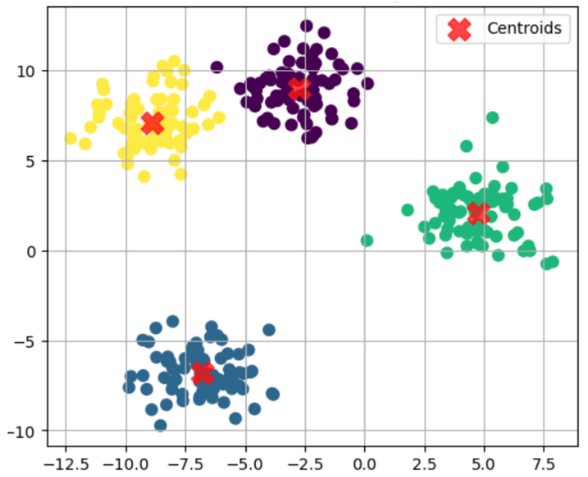
Clustering is one of the most widely used techniques for exploratory data analysis. Across all disciplines, from social sciences to biology to computer science, people try to get a first intuition about their data by **identifying meaningful groups** among the data points.

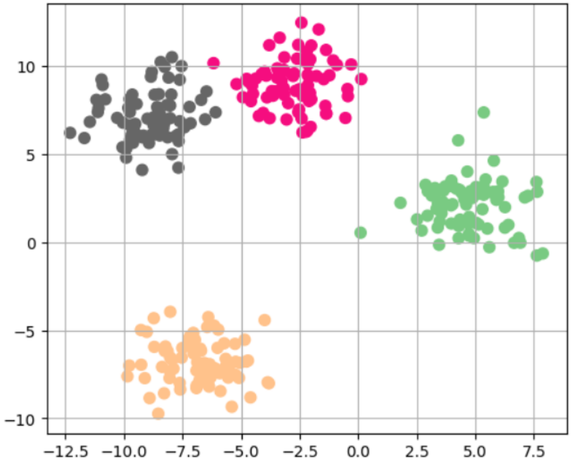
Definition of Clustering

Clustering is a fundamental task in **unsupervised** machine learning. It involves grouping a collection of objects (data points) into subsets or “clusters,” such that data points in the same cluster are more similar (according to some distance or similarity metric) to each other than to those in other clusters.

Unlike supervised learning, where models are trained using labeled data, clustering operates without predefined labels. The goal is to uncover the underlying structure or distribution in the dataset.







Why Do We Cluster?

Clustering is used to:

- Discover natural groupings or patterns in data.
- Reduce the complexity of data.
- Preprocess data for other machine learning tasks such as classification.
- Provide insight or structure in exploratory data analysis.

Feature	Clustering (Unsupervised)	Classification (Supervised)
Input Data	Unlabeled	Labeled (input features + class labels)
Goal	Find structure/groups	Assign data to predefined classes
Output	Cluster assignments (no meaning a priori)	Class labels (with known meanings)
Example	Grouping news articles by topic	Classifying emails as spam or not spam

A Clustering Model

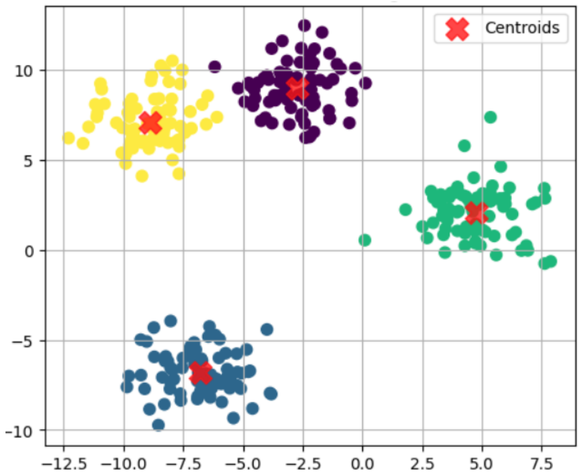
Clustering tasks can vary in terms of both the type of **input** they have and the type of **outcome** they are expected to compute. For concreteness, we shall focus on the following common setup:

Input

A set of elements X and a distance function over it.

That is, a function $d : X \times X \to R_+$ that is symmetric, satisfies $d(x, x) = 0$ for all $x \in X$.

Additionally, some clustering algorithms also require an input parameter k (determining the number of required clusters).



Output

A partition of the domain set X into subsets.

- That is, $C = (C_1, \ldots C_k)$ where $\bigcup_{i=1}^{k} Ci = X$ and for all $i \neq j$, $C_i \cap C_j = \emptyset$.
- The partition of X into the different clusters is probabilistic where the output is a function assigning to each domain point, $x \in X$, a vector $(p_1(x), \ldots, p_k(x))$, where $p_i(x) = P[x \in C_i]$ is the probability that x belongs to cluster C_i .
- The clustering dendrogram which is a hierarchical tree of domain subsets, having the singleton sets in its leaves, and the full domain as its root.

Main types of clustering algorithms

Type of Algorithm	Examples	Main Idea
Centroid-based	K-means, K-median	Objects belong to the cluster with the nearest center (centroid).
Linkage-based (Hierarchical)	Single linkage, Complete linkage, Average linkage, Ward's method	A hierarchy of clusters is constructed by merging groups based on a proximity (similarity) measure.
Density-based	DBSCAN, OPTICS	Clusters are formed as regions of high point density separated by areas of low density ("gaps").
Distribution-based	Gaussian Mixture Models (GMM)	It is assumed that the data are generated from a mixture of several probabilistic distributions.
Graph-based (sometimes considered separately)	Spectral clustering	Clusters are identified based on graph properties (eigenvalues of the adjacency or Laplacian matrix).

K-Means Clustering

A common approach to clustering defines a **cost** (or objective) **function** over a set of possible partitioning.
The goal of the algorithm is to find the partitioning with the **lowest cost**.
In this way, clustering becomes an **optimization problem**, where the objective function maps the input data (X, d) and a proposed clustering solution $C=(C_1,\ldots,C_k)$ to a positive real value.

K-means Objective Function

- The data is divided into *k* disjoint clusters $C_1,\ldots,C_k$, each represented by a centroid μ_i.
- The objective: minimize the sum of squared distances between each point and the centroid of its cluster.

$$G_{k\text{-means}} = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i)^2$$

- Intuition: find centroids $\mu_1,\ldots,\mu_k$ that minimize this distortion.
- Application example: in digital communication, signals can be approximated by their nearest centroid, reducing transmission cost while introducing minimal distortion.

The k-Means Algorithm (Lloyd's algorithm)

- Finding the true optimal solution is computationally challenging — the problem is **NP-hard** (and even NP-hard to approximate within a constant factor).
- Instead, a **simple iterative algorithm** is typically applied.
- In fact, in many contexts, the term k-means clustering refers to the result of this iterative process, rather than the exact global minimum of the k-means cost.
- We will present the algorithm assuming the **Euclidean distance function**: $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ or $d(x, y) = \sqrt{\sum_{k=1}^{d} (x_k - y_k)^2}.$

Input: dataset $X \subset R^n$, number of clusters k

1. **Initialize:** randomly select k centroids $\mu_1, \ldots, \mu_k$.
2. **Repeat until convergence:**
    1. **Assignment step:** for each point $x \in X$, assign it to the cluster with the nearest centroid.
    2. **Update step:** recalculate each centroid μ_i as the mean of all points in its cluster C_i.

input: $\mathcal{X} \subset \mathbb{R}^n$; Number of clusters k
initialize: Randomly choose initial centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
repeat until convergence
$\forall i \in [k]$ set $C_i = \{\mathbf{x} \in \mathcal{X} : i = \operatorname{argmin}_j \|\mathbf{x} - \boldsymbol{\mu}_j\|\}$
(break ties in some arbitrary manner)
$\forall i \in [k]$ update $\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$

Lemma. Each iteration of the k-means algorithm does not increase the k-means objective function

Example of K-means Clustering (manual calculation)

Initial data:

Set of points: X={1,2,7,8}

Number of clusters: k=2

Step 1. Initialization

Choose two initial centroids (for example, the first and last points): $C_1^{(0)} = 1, \quad C_2^{(0)} = 8$

Step 2. First iteration

Assignment:

- Distance from 1: to C1 = 0, to C2 = 7 → assigned to C1
- Distance from 2: to C1 = 1, to C2 = 6 → assigned to C1
- Distance from 7: to C1 = 6, to C2 = 1 → assigned to C2
- Distance from 8: to C1 = 7, to C2 = 0 → assigned to C2

Clusters after assignment:

C1:{1,2},C2:{7,8}

Update centroids: $C_1^{(1)} = \frac{1+2}{2} = 1.5, \quad C_2^{(1)} = \frac{7+8}{2} = 7.5$

Step 3. Second iteration

Assignment with new centroids:

- Distance from 1: to C1 = 0.5, to C2 = 6.5 → C1
- Distance from 2: to C1 = 0.5, to C2 = 5.5 → C1
- Distance from 7: to C1 = 5.5, to C2 = 0.5 → C2
- Distance from 8: to C1 = 6.5, to C2 = 0.5 → C2

Clusters remain the same:

C1:{1,2},C2:{7,8}

Centroids do not change: $C_1^{(2)} = 1.5, \quad C_2^{(2)} = 7.5$

Step 4. Convergence

Since the centroids did not change after the second iteration, the algorithm **converges**.

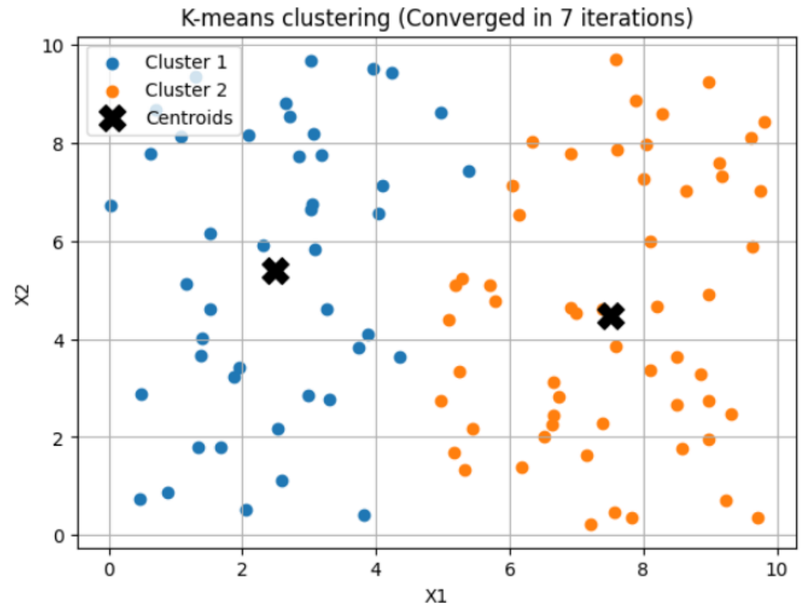
Final clusters: $C_1 : \{1, 2\}, \quad C_2 : \{7, 8\}$

Example of K-means Clustering (Python)

```python
import numpy as np
import matplotlib.pyplot as plt

# === Step 1. Generate 100 random points in R^2 ===
np.random.seed(21)  # for reproducibility
X = np.random.rand(100, 2) * 10  # 100 points in range [0,10)

# === Step 2. K-means implementation ===
def kmeans(X, k=2, max_iters=100, tol=1e-4):
    # Randomly choose initial centroids
    centroids = X[np.random.choice(X.shape[0], k, replace=False)]
    for iteration in range(max_iters):
        # Assign each point to the nearest centroid
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(distances, axis=1)

        # Compute new centroids
        new_centroids = np.array([X[labels == j].mean(axis=0) for j in range(k)])

        # Check for convergence (if centroids do not move more than tol)
        if np.all(np.linalg.norm(new_centroids - centroids, axis=1) < tol):
            return labels, new_centroids, iteration + 1

        centroids = new_centroids

    return labels, centroids, max_iters
```

```python
# Run K-means
labels, centroids, n_iters = kmeans(X, k=2)

print(f"Convergence reached in {n_iters} iterations.")

# === Step 3. Visualization ===
plt.figure(figsize=(7, 5))
for cluster in range(2):
    plt.scatter(X[labels == cluster, 0], X[labels == cluster, 1], label=f"Cluster {cluster+1}")
plt.scatter(centroids[:, 0], centroids[:, 1], c="black", marker="X", s=200, label="Centroids")
plt.title(f"K-means clustering (Converged in {n_iters} iterations)")
plt.xlabel("X1")
plt.ylabel("X2")
plt.legend()
plt.grid(True)
plt.show()
```

```
Convergence reached in 7 iterations.
```



sklearn.cluster.KMeans

Category	Name	Type / Default Value	Description (Academic Style)
Parameters	n_clusters	int, default=8	Defines the number of clusters to form and the number of centroids to generate.
	init	{'k-means++', 'random'} or ndarray, default='k-means++'	Specifies the method for initializing the centroids. The 'k-means++' method accelerates convergence.
	n_init	int or 'auto', default=10	Determines the number of times the k-means algorithm will be run with different centroid seeds; the best result is selected based on the inertia criterion.
	max_iter	int, default=300	Maximum number of iterations of the algorithm for a single run.
	tol	float, default=1e-4	Relative tolerance with respect to inertia to declare convergence.
	random_state	int, RandomState instance or None, default=None	Ensures reproducibility of results when the initialization is random.
	algorithm	{'lloyd', 'elkan'}, default='lloyd'	Specifies the algorithm to use: Lloyd (standard) or Elkan (more efficient for dense datasets).
	copy_x	bool, default=True	Determines whether the input data is copied or modified in place during the computation.
	n_jobs	None or int, default=None (deprecated)	Specifies the number of parallel jobs to run for computation.
	init_size	int or None, optional	Determines the number of samples to randomly select for centroid initialization when using mini-batch variants.

sklearn.cluster.KMeans

Category	Name	Type / Default Value	Description (Academic Style)
Attributes	cluster_centers_	ndarray of shape (n_clusters, n_features)	Coordinates of the cluster centers after training.
	labels_	ndarray of shape (n_samples,)	Labels of each point, indicating the assigned cluster.
	inertia_	float	Sum of squared distances of samples to their closest cluster center — used as a measure of internal cluster compactness.
	n_iter_	int	Number of iterations performed during the final run.
	n_features_in_	int	Number of features seen during fitting.
	feature_names_in_	ndarray of shape (n_features_in_,)	Names of features seen during fitting (if input is a DataFrame).

Choosing the Number of Clusters

Determining the optimal number of clusters, K, is an important task in K-means clustering.

Elbow method

Idea: Analyze the reduction in the sum of squared errors (SSE) with increasing number of clusters.

$$SSE(k) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where C_i is the i-th cluster and μ_i its centroid

Algorithm:

- For k=1,2,…,K_{max}: run clustering and compute $SSE(k)$.
- Plot $SSE(k)$ vs k.
- Choose k at the “elbow” point — where the decrease in SSE slows down.

Silhouette analysis

Idea: Measure how well objects are clustered compared to neighboring clusters.

For object i:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where

$a(i)$ = average distance from i to all other points in the same cluster,

$b(i)$ = minimum average distance from i to points in other clusters.

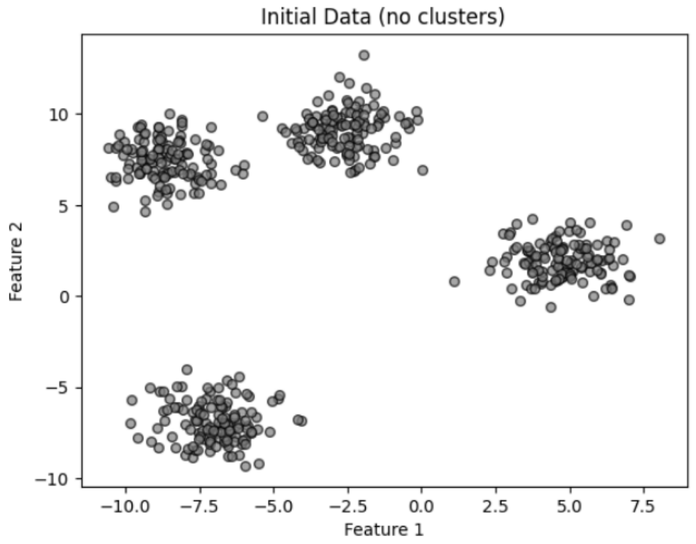
The overall silhouette score (Mean Silhouette Score):

$$S(k) = \frac{1}{n} \sum_{i=1}^{n} s(i)$$

Algorithm:

- For k=2,3,…,K_{max}: run clustering.
- Compute silhouette score $S(k)$.
- Choose k that maximizes $S(k)$.

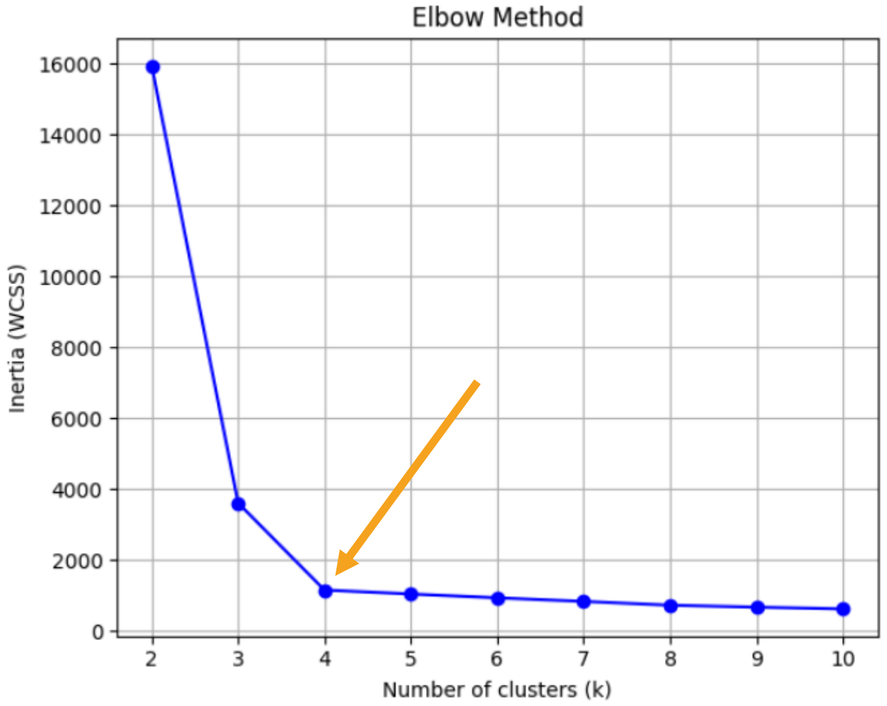
Example. Choosing the Number of Clusters

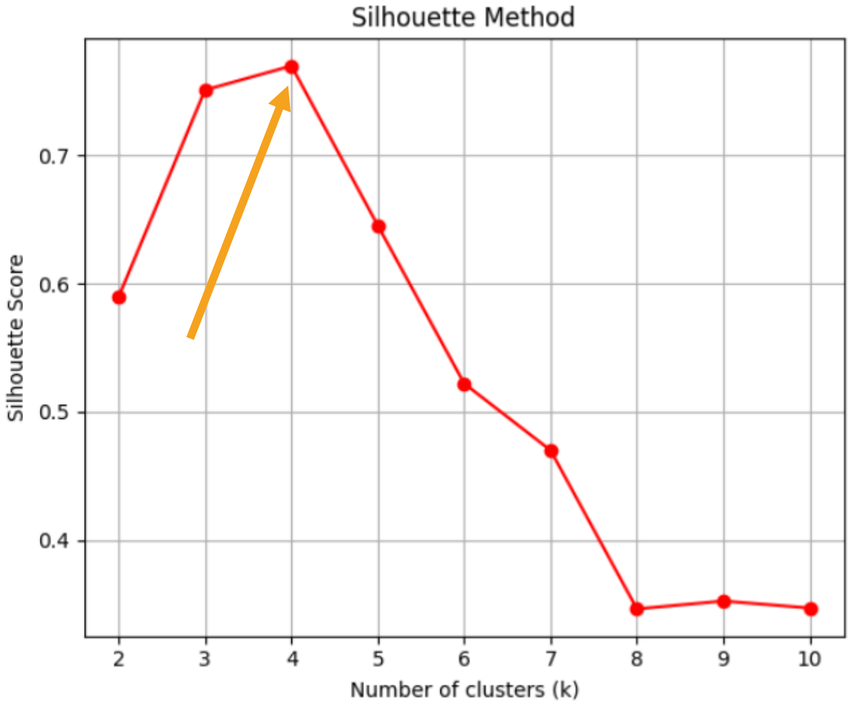



```python
# === 2. Elbow Method ===
inertia = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(K_range, inertia, 'bo-')
plt.xlabel("Number of clusters (k)")
plt.ylabel("Inertia (WCSS)")
plt.title("Elbow Method")
plt.grid(True)
```

```python
# === 3. Silhouette Method ===
silhouette_scores = []

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X)
    score = silhouette_score(X, labels)
    silhouette_scores.append(score)

plt.subplot(1, 2, 2)
plt.plot(K_range, silhouette_scores, 'ro-')
plt.xlabel("Number of clusters (k)")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Method")
plt.grid(True)

plt.tight_layout()
plt.show()
```

Number of clusters K=4





Code Exercise (K-means Clustering)

Task 1 (Manual Calculation)
You are given the following 6 two-dimensional points:
(1,1),(2,1),(4,3),(5,4),(6,5),(8,7)
Perform **one full iteration** of the **k-means algorithm** with k=2:
1.Choose initial centroids as C1=(1,1) and C2=(8,7).
2.Assign each point to the closest centroid (Euclidean distance).
3.Compute the new centroids of the formed clusters.
Show all intermediate calculations (distances, assignments, centroid updates).

Task 2 (Python Implementation)
Implement **k-means clustering in Python from scratch** for a 2D dataset of at least 10 points.
Steps to include in your code:
- Randomly initialize centroids.
- Assign each point to the nearest centroid.
- Recompute centroids.
- Repeat until centroids do not change significantly.
- Plot the points and cluster centroids using **matplotlib**.

Task 3 (Comparison with `sklearn`)
Generate a synthetic dataset using `make_blobs` from `sklearn.datasets` with three clusters.
Apply:
- Your own custom implementation of k-means from Task 2.
- The built-in `KMeans` from `sklearn`.

Compare:
- The final centroids.
- The number of iterations until convergence.
- The visualization of clusters.

Link to the task:

https://colab.research.google.com/drive/1kXymA2FGd1-WJgVYSmjXrwdo5FxQKXsY?usp=sharing

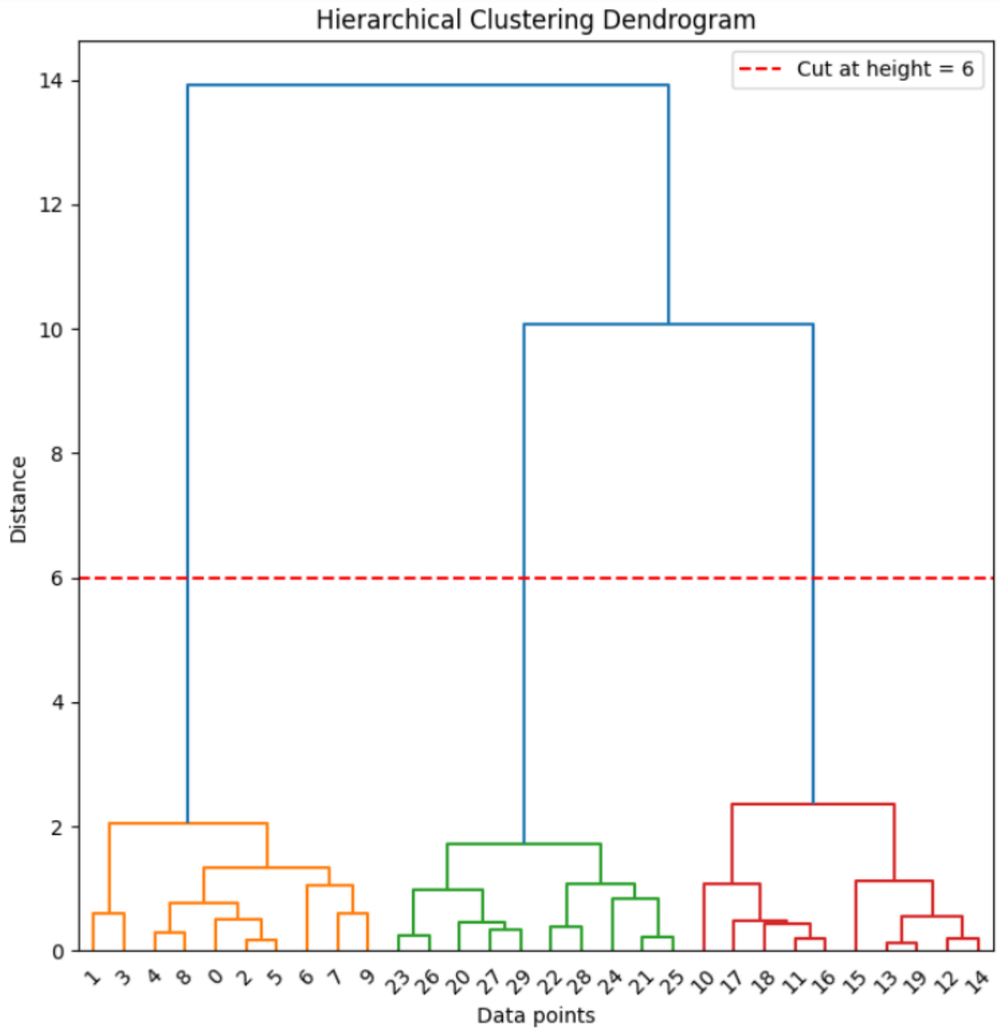
General Idea of Hierarchical Clustering

Hierarchical clustering is a method of cluster analysis that builds a hierarchy of clusters rather than producing just a single partition of the data. Unlike **k-means**, which requires us to ***predefine the number of clusters***, hierarchical clustering can reveal the structure of the data at different levels of granularity.

- Agglomerative (bottom-up): Each observation starts as its own cluster, and pairs of clusters are merged step by step until all data points belong to one single cluster.
- Divisive (top-down): All data points start in one cluster, and the cluster is recursively split into smaller clusters until each observation is separate.

In practice, agglomerative clustering is most commonly used because it is conceptually simpler and easier to compute.

The result of hierarchical clustering is often represented as a ***dendrogram*** – a tree-like diagram that shows how clusters are merged (or divided) at each step.
By "cutting" the dendrogram at a chosen level, we can decide how many clusters to form.



Definition of $|C_i|$

In **hierarchical clustering**, when we cut the dendrogram at a chosen height, we obtain a set of clusters.

Let's denote them as: $C=\{C_1, C_2, \ldots, C_k\}$,

- k is the number of clusters formed after the cut.
- C_i represents the i-th cluster.
- $|C_i|$ denotes the cardinality (size) of cluster C_i.

In other words, $|C_i|$ is simply the number of data points (observations) that belong to the cluster C_i.

Example

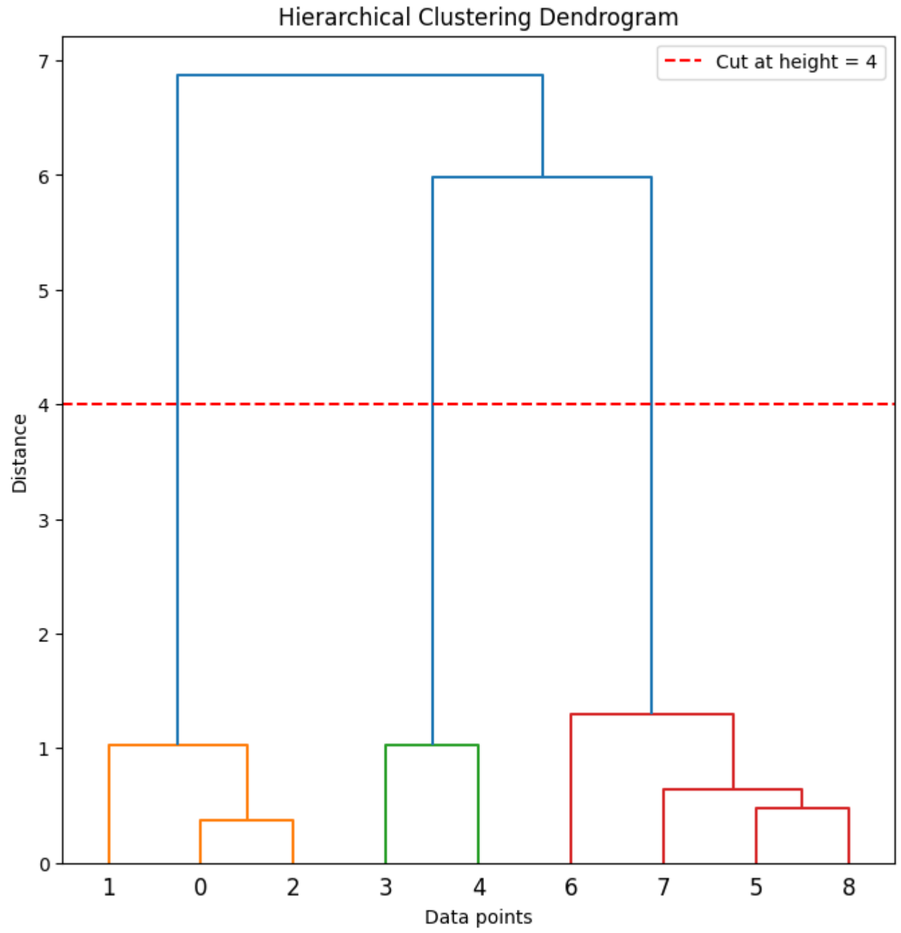
Suppose we cut the dendrogram at some level and obtain 3 clusters:

$$C_1 = \{x_1, x_4, x_7\}, \quad C_2 = \{x_2, x_3\}, \quad C_3 = \{x_5, x_6, x_8, x_9\}$$

- $|C_1| = 3$
- $|C_2| = 2$
- $|C_3| = 4$

So the total number of points is preserved: $\sum_{i=1}^{k} |C_i| = n$

where n is the total number of observations in the dataset.



Agglomerative Hierarchical Clustering – Step-by-Step

- Initialization: Start with each data point as its own cluster.
- Compute Distances: Calculate the distance between every pair of clusters (initially between individual points).
- Merge Closest Clusters: Find the two clusters with the smallest distance and merge them into a new cluster.
- Update Distances: Recalculate the distances between the new cluster and all remaining clusters.
- Repeat: Continue steps 3-4 until all points are merged into one cluster.

The **key factor** is how we define the "**distance**" between clusters. Common linkage methods include:

- Single linkage: distance between the closest members of two clusters.
- Complete linkage: distance between the farthest members of two clusters.
- Average linkage: average distance between all members of two clusters.
- Ward's method: minimizes the total within-cluster variance (often preferred in practice).

Let us consider two sets of points:

$$C_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}, \quad C_j = \{x_{j1}, x_{j2}, \ldots, x_{jn}\}, \quad x_{ik},\ x_{jl} \in \mathbb{R}^d.$$

The distance between two points: $d(x,y) = \sqrt{\sum_{k=1}^{d}(x_k - y_k)^2}.$

Single Linkage. We take the closest pair of points between two clusters:

$$D_{\text{single}}(C_i, C_j) = \min_{x \in C_i,\, y \in C_j} d(x,y).$$

Complete Linkage. We take the farthest pair of points between two clusters:

$$D_{\text{complete}}(C_i, C_j) = \max_{x \in C_i,\, y \in C_j} d(x,y).$$

Average Linkage. We take the average distance between all pairs of points:

$$D_{\text{average}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x,y).$$

Ward's Method. Merge clusters that minimize the increase in total within-cluster variance:

$$D_{\text{Ward}}(C_i, C_j) = \frac{|C_i| \cdot |C_j|}{|C_i| + |C_j|} \|\bar{x}_i - \bar{x}_j\|^2,$$

where $\bar{x}_i$ and $\bar{x}_j$ are the centroids (mean vectors) of clusters Ci and Cj

Example. Agglomerative Hierarchical Clustering using Single Linkage

Given X={1,2,3,8,9}

We'll perform clustering "by hands", showing all steps: initial distances, merging decisions, updating the distance matrix using single linkage:

Step 1: Understanding the Setup

Agglomerative Hierarchical Clustering:

- Starts with each point as its own cluster.
- Iteratively merges the two closest clusters.
- Continues until all points are in one cluster.

Single Linkage:

- Distance between two clusters $D_{\text{single}}(C_i, C_j) = \min_{x \in C_i,\, y \in C_j} d(x, y).$
- That is, the minimum distance between any two points, one from each cluster.
- Data as 1D points, so distance is absolute difference:

$$d(x_i, x_j) = |x_i - x_j|$$

Step 2: Initial Clusters

Each point is its own cluster:

$$C_1 = \{1\},\ C_2 = \{2\},\ C_3 = \{3\},\ C_4 = \{8\},\ C_5 = \{9\}$$

Step 3: Compute Initial Distance Matrix

We compute pairwise distances between clusters (each singleton now):

	{1}	{2}	{3}	{8}	{9}
{1}	0	1	2	7	8
{2}	1	0	1	6	7
{3}	2	1	0	5	6
{8}	7	6	5	0	1
{9}	8	7	6	1	0

We only need the lower triangle. Let's find the minimum non-zero distance.

Pairs with distance 1:
{1}-{2}, {2}-{3}, {8}-{9}
We can pick any of them. Let's pick the leftmost pair:
{1} and {2}.

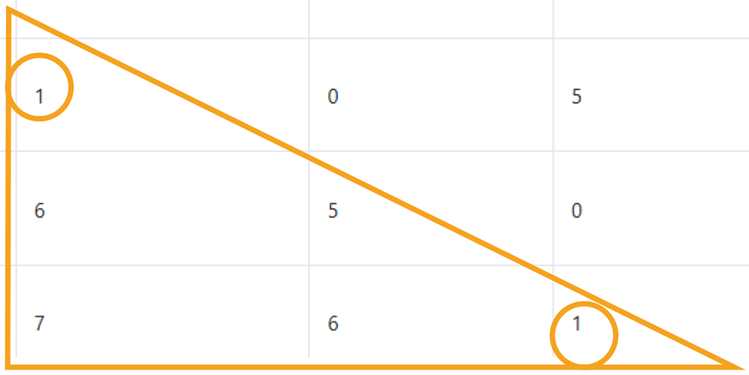
Example. Agglomerative Hierarchical Clustering using Single Linkage

Step 4: Merge {1} and {2} → New Cluster {1,2}

Now clusters are:
{1,2}, {3}, {8}, {9}

Update distance matrix using single linkage:
Compute distances from {1,2} to others:
d({1,2},{3})=min(|1−3|,|2−3|)=min(2,1)=1
d({1,2},{8})=min(|1−8|,|2−8|)=min(7,6)=6
d({1,2},{9})=min(|1−9|,|2−9|)=min(8,7)=7
Other distances remain the same.
New distance matrix:

	{1,2}	{3}	{8}	{9}
{1,2}	0	1	6	7
{3}	1	0	5	6
{8}	6	5	0	1
{9}	7	6	1	0



Minimum distance = 1 Pairs: {1,2}-{3}, {8}-{9}
We can merge either. Let's merge {1,2} and {3} → {1,2,3}

Step 5: Merge {1,2} and {3} → {1,2,3}

Now clusters: {1,2,3}, {8}, {9}

Update distances:
d({1,2,3},{8})=min(|1−8|,|2−8|,|3−8|)=min(7,6,5)=5
d({1,2,3},{9})=min(|1−9|,|2−9|,|3−9|)=min(8,7,6)=6
Distance between {8} and {9}: |8−9|=1
New matrix:

	{1,2,3}	{8}	{9}
{1,2,3}	0	5	6
{8}	5	0	1
{9}	6	1	0

Minimum distance = 1 → between {8} and {9}
Merge {8} and {9} → {8,9}

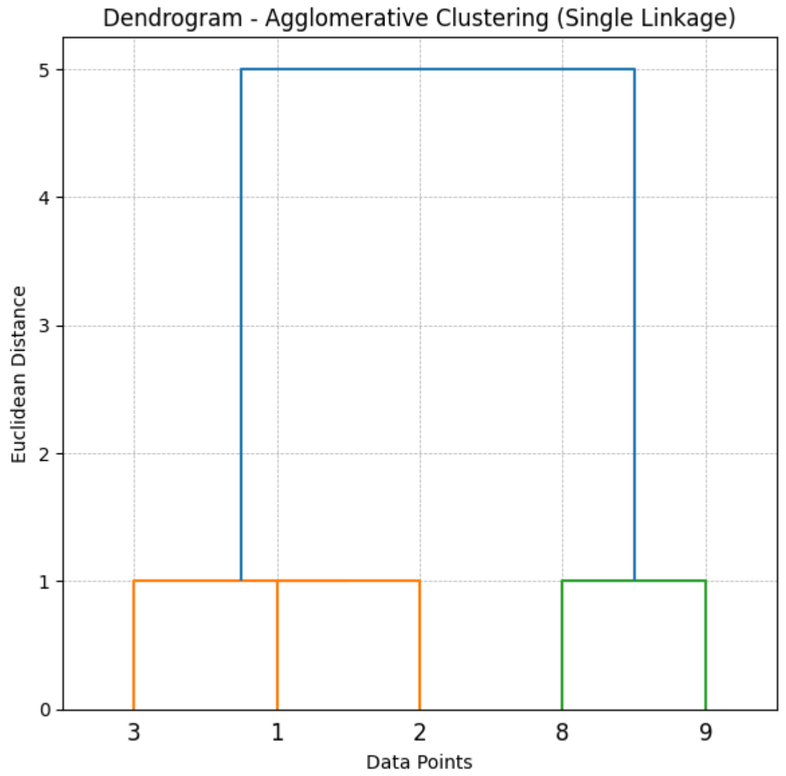
Example. Agglomerative Hierarchical Clustering using Single Linkage

Step 6: Merge {8} and {9} → {8,9}
Now clusters: {1,2,3}, {8,9}

Compute distance between them:

d({1,2,3},{8,9})=min(|x−y| for x∈{1,2,3},y∈{8,9})
=min(|1−8|,|1−9|,|2−8|,|2−9|,|3−8|,|3−9|)
=min(7,8,6,7,5,6)=5
So distance = 5

	{1,2,3}	{8,9}
{1,2,3}	0	5
{8,9}	5	0



Step 7: Final Merge
Only two clusters left: {1,2,3} and {8,9}, distance = 5
merge them.
Final cluster: {1,2,3,8,9}

Conclusion
Using single linkage, the clustering naturally separates the data into two natural groups:
Left cluster: {1,2,3} (close together)
Right cluster: {8,9} (close together)
With a large gap (from 3 to 8) of 5 units, which is the final merge distance.

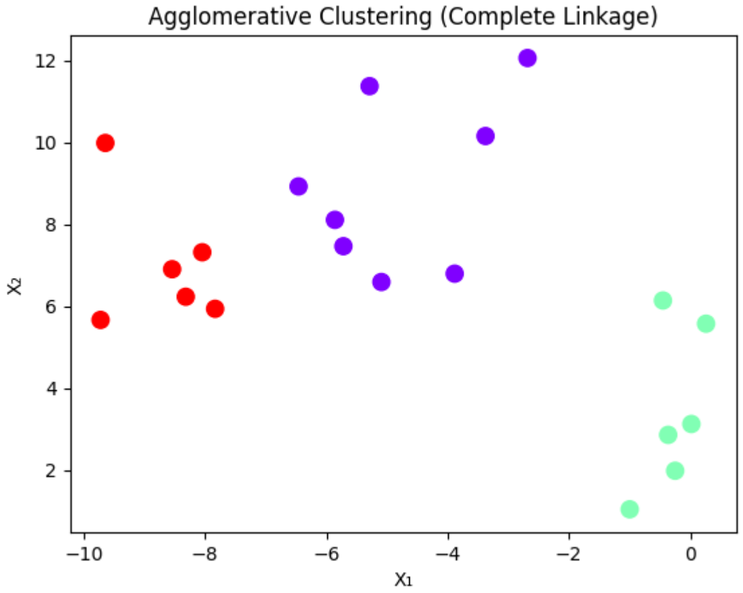
Python - `sklearn.cluster.AgglomerativeClustering`

Parameter / Attribute	Type / Options	Description
n_clusters	int (default=2)	The number of clusters to form after the merging process.
metric	str or callable (available from sklearn ≥ 1.2)	The distance metric to use for computing pairwise distances between samples. Default is 'euclidean'.
linkage	'ward', 'complete', 'average', 'single'	The linkage criterion that determines the distance between clusters: • 'ward' — minimizes within-cluster variance (only compatible with euclidean), • 'complete' — maximum pairwise distance, • 'average' — average pairwise distance, • 'single' — minimum pairwise distance.
compute_full_tree	bool or 'auto'	If True, builds the full linkage tree; if False, stops early once the desired number of clusters is reached.
distance_threshold	float or None	If set, clustering stops at a specified distance threshold, and n_clusters is ignored.
compute_distances	bool	If True, stores the distances between merged clusters (useful for dendrogram construction).
Class methods		
.fit(X)	—	Fits the hierarchical clustering model to the dataset X.
.fit_predict(X)	—	Performs model fitting and returns cluster labels for each observation.
.labels_	ndarray	Cluster labels for each sample after fitting.
.n_clusters_	int	The number of clusters found by the algorithm.
.distances_	ndarray	Distances between clusters merged at each step (available if compute_distances=True).

Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

# --- Генеруємо штучні дані ---
X, _ = make_blobs(n_samples=20, n_features=2, centers=3, cluster_std=2.5, random_state=5
print(X)
# --- Ієрархічна кластеризація ---
model = AgglomerativeClustering(n_clusters=3, linkage='complete')
labels = model.fit_predict(X)

# --- Візуалізація ---
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='rainbow', s=80)
plt.title("Agglomerative Clustering (Complete Linkage)")
plt.xlabel("X₁")
plt.ylabel("X₂")
plt.show()

print("Мітки кластерів:", labels)
```

```
[[ 0.02006925  3.12347344]
 [-0.3625466   2.8579214 ]
 [-0.44896524  6.13870799]
 [-7.83321759  5.93555448]
 [-8.54204811  6.90245485]
 [-0.99640878  1.04054871]
 [-5.09112851  6.58997123]
 [-0.2472635   1.98220823]
 [-6.45720895  8.92332513]
 [-2.68165905 12.05797364]
 [-8.04531485  7.31594834]
 [-3.88298359  6.79328908]
 [-5.71775629  7.46394096]
 [-9.7221079   5.66419853]
 [-5.85739479  8.10739205]
 [-3.37451733 10.15327134]
 [-8.31713661  6.23008527]
 [-5.28611197 11.37084892]
 [ 0.262374    5.57199869]
 [-9.64356579  9.98433694]]
```




```
Мітки кластерів: [1 1 1 2 2 1 0 1 0 0 2 0 0 2 0 0 2 0 1 2]
```

Python - `scipy.cluster.hierarchy.linkage`

Parameter / Attribute	Type / Options	Description (in English)
y	ndarray or array of observations	Input data. Can be either: • an observation matrix of size (n × m); • a condensed (flattened) distance matrix representing pairwise distances between points.
method	str	The linkage method used to compute distances between clusters: • 'single' — minimum distance (produces a chaining effect); • 'complete' — maximum distance (forms compact clusters); • 'average' — average pairwise distance; • 'ward' — minimizes within-cluster variance; • 'centroid', 'median', 'weighted' — alternative variants.
metric	str	The distance metric to compute pairwise distances between points, e.g. 'euclidean', 'cityblock', 'cosine', 'chebyshev', etc.
optimal_ordering	bool (default=False)	If True, the leaf order in the dendrogram is optimized to improve the visual layout and interpretability.
Output	ndarray (linkage matrix)	Returns an array of shape (n−1) × 4, where each row describes a merging step: 1 index of the first cluster, 2 index of the second cluster, 3 distance between them, 4 number of observations in the newly formed cluster.

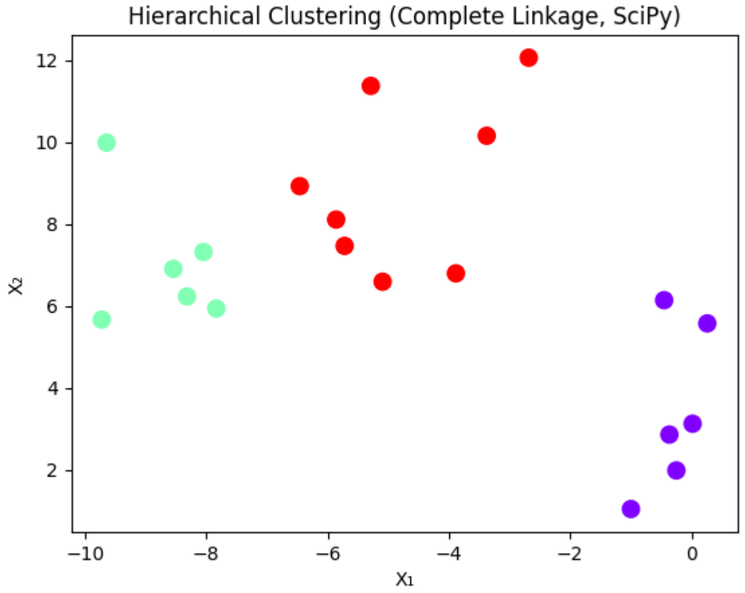
Python - `scipy.cluster.hierarchy.dendrogram`

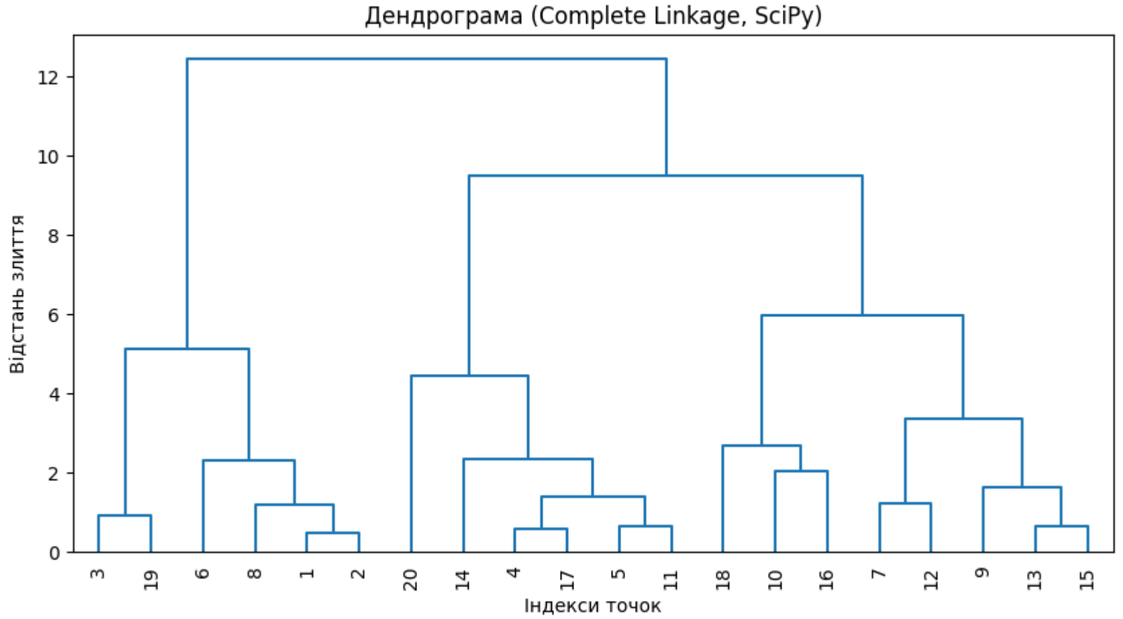
Parameter / Attribute	Type / Options	Description
Z	ndarray	The linkage matrix produced by the linkage() function.
p	int or None	The number of clusters or the truncation level of the dendrogram. Used together with the truncate_mode parameter.
truncate_mode	'lastp', 'level', or None	Defines how to truncate the dendrogram: • 'lastp' — shows only the last p merges; • 'level' — displays the tree up to a specified depth.
labels	list or ndarray	Labels (names) of the leaves — useful for displaying actual point values.
color_threshold	float or None	The distance threshold at which to apply color differentiation between clusters.
leaf_rotation	float	The rotation angle of the leaf labels (for better readability).
leaf_font_size	float	The font size of the leaf labels.
orientation	'top', 'bottom', 'left', 'right'	Orientation of the dendrogram.
no_labels	bool	If True, leaf labels are not displayed.
Return value	dict	A dictionary containing coordinates of branches, colors, and node identifiers — useful for visualization and further analysis.

Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

# --- Генеруємо штучні дані ---
X, _ = make_blobs(n_samples=20, n_features=2, centers=3, cluster_std=2.5, random_state=5)
print("Масив X:\n", X)

# --- Обчислення матриці зв’язків (linkage matrix) ---
Z = linkage(X, method='complete', metric='euclidean')

# --- Побудова дендрограми ---
plt.figure(figsize=(10, 5))
dendrogram(Z,
           labels=np.arange(1, len(X)+1),  # нумерація точок
           leaf_rotation=90,
           leaf_font_size=10,
           color_threshold=0)
plt.title("Дендрограма (Complete Linkage, SciPy)")
plt.xlabel("Індекси точок")
plt.ylabel("Відстань злиття")
plt.show()

# --- Виділення кластерів (наприклад, 3 кластери) ---
labels = fcluster(Z, t=3, criterion='maxclust')
print("Мітки кластерів:", labels)

# --- Візуалізація точок з кольорами кластерів ---
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='rainbow', s=80)
plt.title("Hierarchical Clustering (Complete Linkage, SciPy)")
plt.xlabel("X₁")
plt.ylabel("X₂")
plt.show()
```

```
Масив X:
 [[ 0.02006925  3.12347344]
 [-0.3625466   2.8579214 ]
 [-0.44896524  6.13870799]
 [-7.83321759  5.93555448]
 [-8.54204811  6.90245485]
 [-0.99640878  1.04054871]
 [-5.09112851  6.58997123]
 [-0.2472635   1.98220823]
 [-6.45720895  8.92332513]
 [-2.68165905 12.05797364]
 [-8.04531485  7.31594834]
 [-3.88298359  6.79328908]
 [-5.71775629  7.46394096]
 [-9.7221079   5.66419853]
 [-5.85739479  8.10739205]
 [-3.37451733 10.15327134]
 [-8.31713661  6.23008527]
 [-5.28611197 11.37084892]
 [ 0.262374    5.57199869]
 [-9.64356579  9.98433694]]
```






```
Мітки кластерів: [1 1 1 2 2 1 3 1 3 3 2 3 3 2 3 3 2 3 1 2]
```

`sklearn.cluster.AgglomerativeClustering` VS `scipy.cluster.hierarchy`

Feature	scipy.cluster.hierarchy	sklearn.cluster.AgglomerativeClustering
Output	Dendrogram, linkage (merge) matrix	Cluster labels only
Tree visualization	Available (via dendrogram)	Not directly available (can be reconstructed using `distances_`)
Ease of use	More flexible for analytical exploration	Simpler for practical clustering tasks
Orientation	Focused on analyzing cluster structure	Focused on automatic cluster assignment
Integration in ML pipelines	Limited	Fully integrated with scikit-learn

Comparative Table for the Use of Metrics

Metric	Purpose	Applicable Clustering Methods	Library	Method / Function	Main Parameters	Comment
Inertia	Measures cluster compactness — the sum of squared distances from all points to their nearest centroid.	✅ K-Means	scikit-learn	KMeans.inertia_ (attribute after fit)	-	Used to determine the optimal number of clusters (elbow method).
Silhouette Coefficient (per sample)	Evaluates how well each data point fits within its assigned cluster (inter-cluster quality).	✅ K-Means, ✅ Agglomerative Clustering, ✅ DBSCAN, ✅ SOM	scikit-learn	silhouette_samples(X, labels)	metric='euclidean' (default), alternatives: 'manhattan', 'cosine'	Returns an array of silhouette values for individual samples.
Mean Silhouette Score	Provides an overall assessment of clustering quality (average of individual silhouette scores).	✅ K-Means, ✅ Agglomerative Clustering, ✅ DBSCAN	scikit-learn	silhouette_score(X, labels)	metric='euclidean', sample_size=None, random_state=None	The score ranges from -1 to 1; values closer to 1 indicate better-defined clusters.

Task 1 (Manual Calculation)

You are given the following one-dimensional data points:

X={2,5,8,11,16}

Perform Agglomerative Hierarchical Clustering using Single Linkage (minimum distance between clusters) step by step, as we did in class.

1.Start with each point as its own cluster.

2.Compute the initial pairwise distance matrix (using absolute difference).

3.At each step:

- Identify the two closest clusters.
- Merge them into a new cluster.
- Update the distance matrix using single linkage.

4.Continue until all points are in one cluster.

Deliverables:

•Show the distance matrix at each step.

•Clearly state which clusters are merged and at what distance.

•Draw or describe the resulting dendrogram (label merge heights).

•Based on the dendrogram, list the clusters if you were to choose 2 final clusters.

Task 2 (Python Coding - Basic Implementation)

Write a Python script to perform Agglomerative Hierarchical Clustering on the dataset:

X=[2,5,8,11,16]

Instructions:

Use `scipy.cluster.hierarchy` to:

1. Compute the linkage matrix using single linkage and Euclidean distance.
    - Plot a clear dendrogram with labeled leaves (show the actual values: 2, 5, 8, 11, 16).
    - Set appropriate labels and title.
2. Repeat the process using complete linkage.
3. Display both dendrograms side by side for comparison.

Deliverables:

- Your full Python code.
- The two dendrogram plots.
- A short written comparison (2–3 sentences): How do the clustering structures differ? Which method results in higher merge distances?

Let’s proceed to the practical exercises

link:
https://colab.research.google.com/drive/10cJAbEwZOc4oFMYgfSS38fGMXRJS2ZhO?usp=sharing

REFERENCES

1. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in R* (2nd ed.). Springer. https://doi.org/10.1007/978-1-0716-1418-1
2. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer. https://doi.org/10.1007/978-0-387-84858-7