CLUSTERING ALGORITHMS

Machine Learning – Unsupervised Learning Type

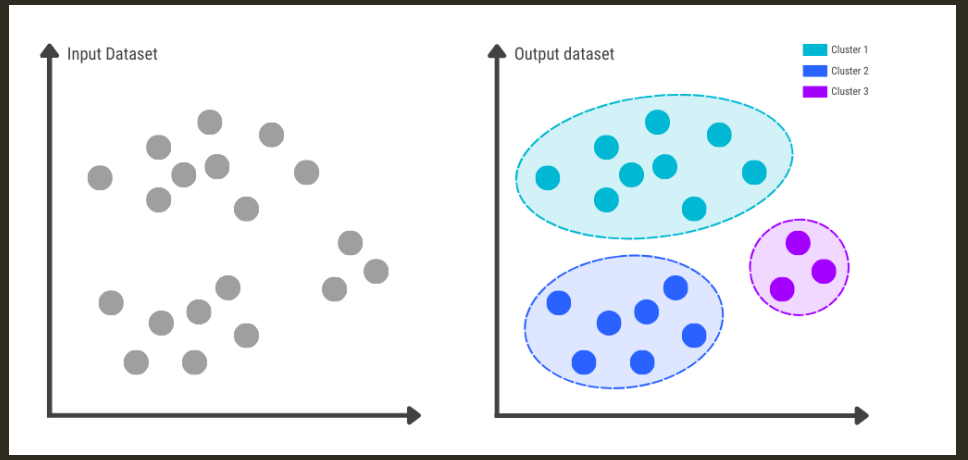
CLUSTERING

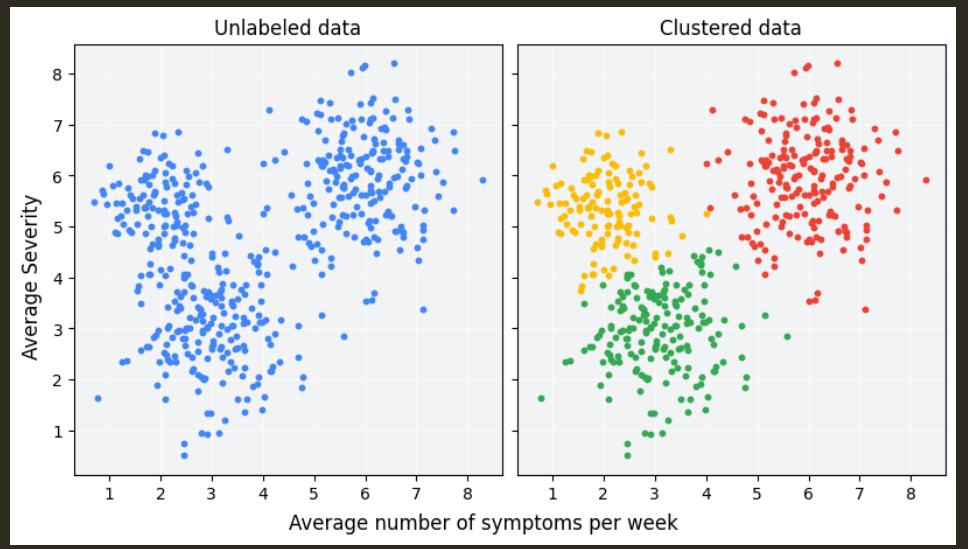
Clustering is an **unsupervised machine learning technique** used to **group similar data points together.**

- The main idea: **Data points in the same cluster are more similar to each other than to points in other clusters.**
- There are **no labels** in unsupervised learning; the algorithm discovers structure in the data itself.

Use cases in real life:

- Customer segmentation (marketing)
- Document grouping (NLP)
- Anomaly detection (fraud, manufacturing defects)
- Image segmentation

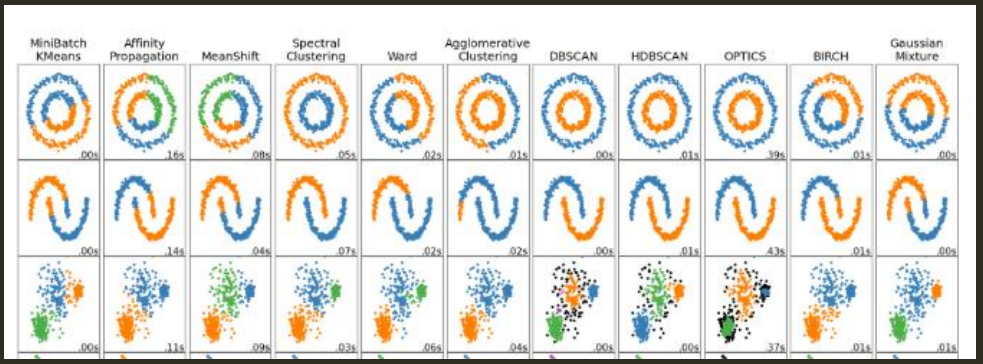


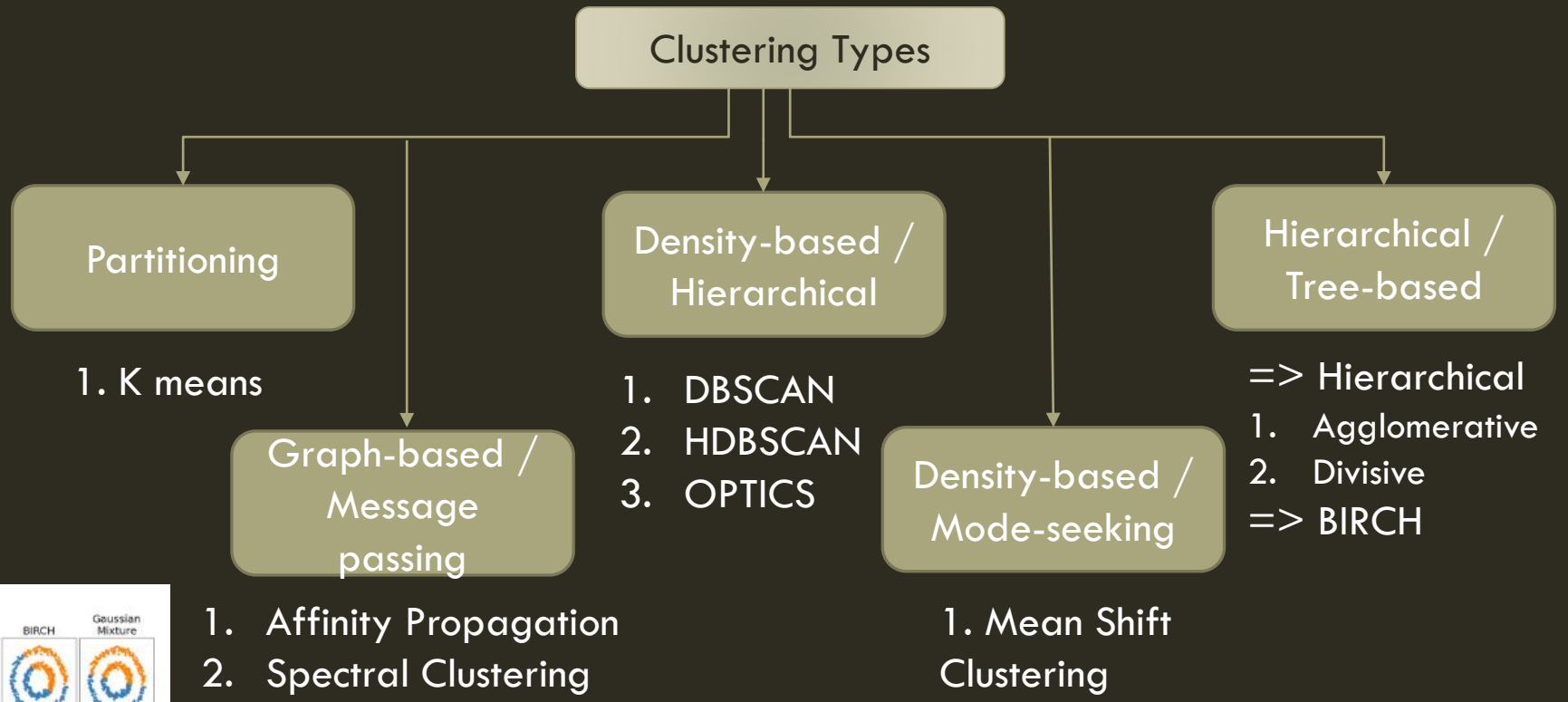


KEY CHALLENGES IN CLUSTERING & IT'S TYPES

KEY CHALLENGES:-

- **Number of Clusters Unknown**
- **Irregular Shapes**
- **High-Dimensional Data**
- **Outliers and Noise**



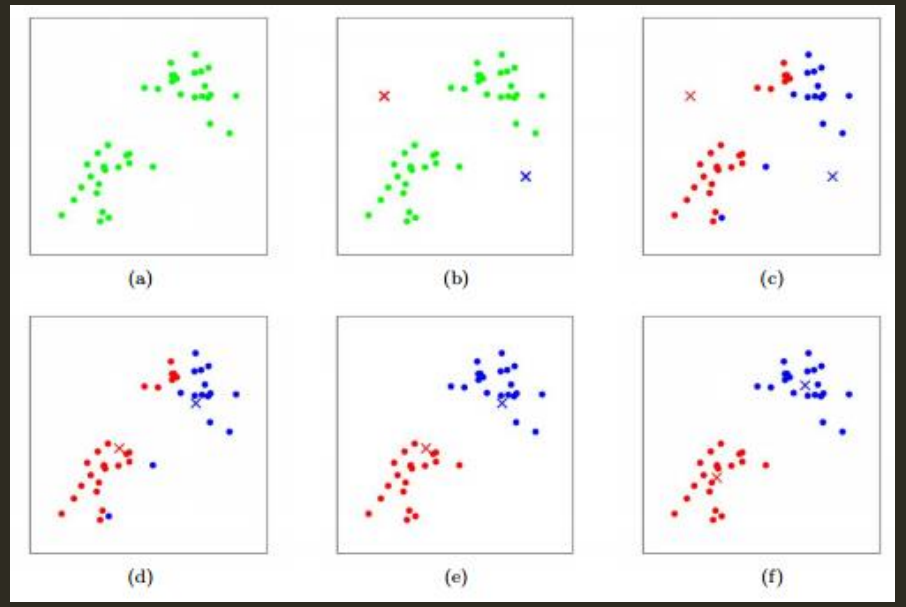


PARTITIONING

K MEANS = K + CENTROID + ITERATE UNTIL CONVERGENCE

Partition-based algorithm that divides data into **k clusters** by minimizing **within-cluster variance (SSE)**. Cluster centers are **centroids (mean of points)**.

- **Type:** Unsupervised clustering algorithm
- **Purpose:** Automatically groups similar data points into **K clusters**



Working Steps:

1. Initialize k centroids randomly (or k-means++)
2. Assign each point to nearest centroid
3. Update centroid = mean of assigned points
4. Repeat steps 2–3 until centroids don't change or max iterations reached

Advantage: simple, fast and Scales well for large datasets
Disadvantage: specify K, only for convex/spherical clusters
Application: customer segmentation, document clustering

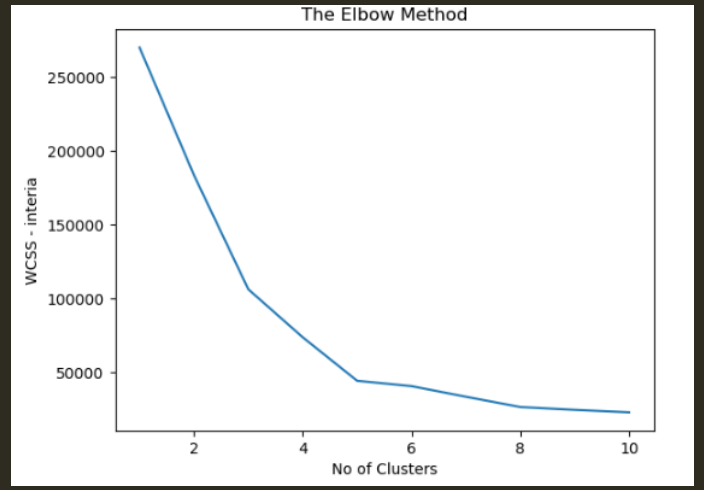
CODING & REAL WORLD EXAMPLE

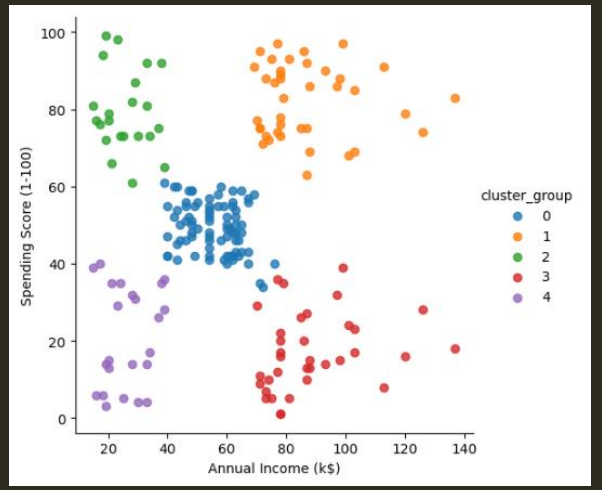
```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

list_Kmeaninteria = []
for i in range(1,11):
    Kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    Kmeans.fit(x) #model creation
    list_Kmeaninteria.append(Kmeans.inertia_)   #Sum of squared distances of samples to their closest cluster center
plt.plot(range(1,11), list_Kmeaninteria)   # (no.of clusters, interia)
plt.title("The Elbow Method")
plt.xlabel("No of Clusters")
plt.ylabel("WCSS - interia")
plt.show()
```

Model Creation

```python
from sklearn.cluster import KMeans
Kmeans_model = KMeans(n_clusters=5, init='k-means++', random_state=42)
y_mean_predicted = Kmeans_model.fit_predict(x)       # model creation
y_mean_predicted
```

```
array([4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4,
       4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 0,
       4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1, 0, 1, 3, 1, 3, 1,
       0, 1, 3, 1, 3, 1, 3, 1, 3, 1, 0, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1], dtype=int32)
```

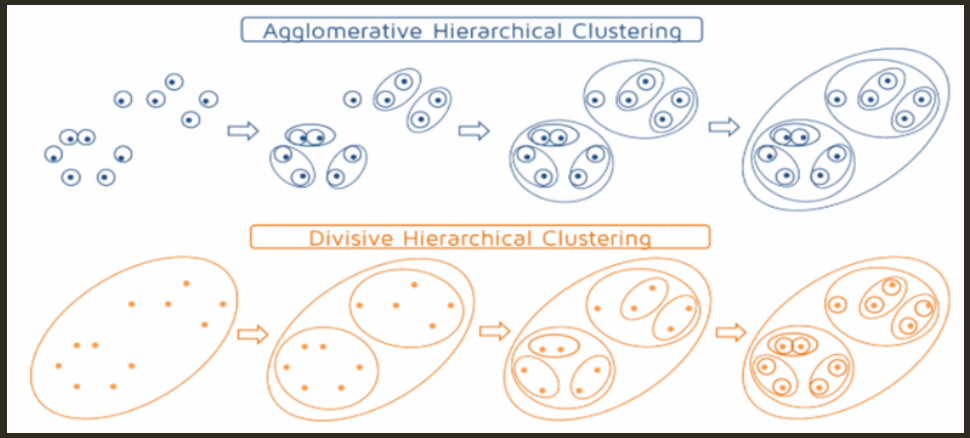





```python
# ANALYSEZ FROM THE ABOVE GRAPH
"""
cluster group category    |    salary     |  spending   |   Saving    |   result
          0                    $ 40-80          40-60%        40-60%        right customers to target
          1                    $ 70-140         20-40%        60-80%        right customers to target
          2                    $ 20-45          20-40%        60-80%        right customers to target
          3                    $ 20-50          60-100%       0%            customers with less savings
          4                    $ 70-140         60-100%       0%            customers with less savings

"""
```

HIERARCHICAL / TREE-BASED CLUSTERING TYPES

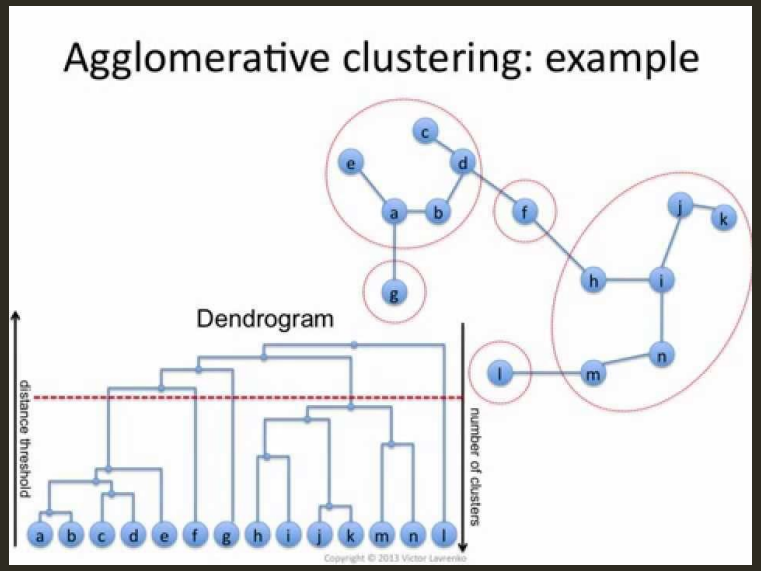
HIERARCHICAL CLUSTERING



1. Agglomerative (mostly used)
2. Divisive(rarely used – due to high computational)

Agglomerative = bottom-up + merge closest clusters + dendrogram

A **bottom-up hierarchical clustering** algorithm where each point starts as its own cluster and clusters are **merged iteratively** based on similarity until all points form one cluster or a stopping condition is met.



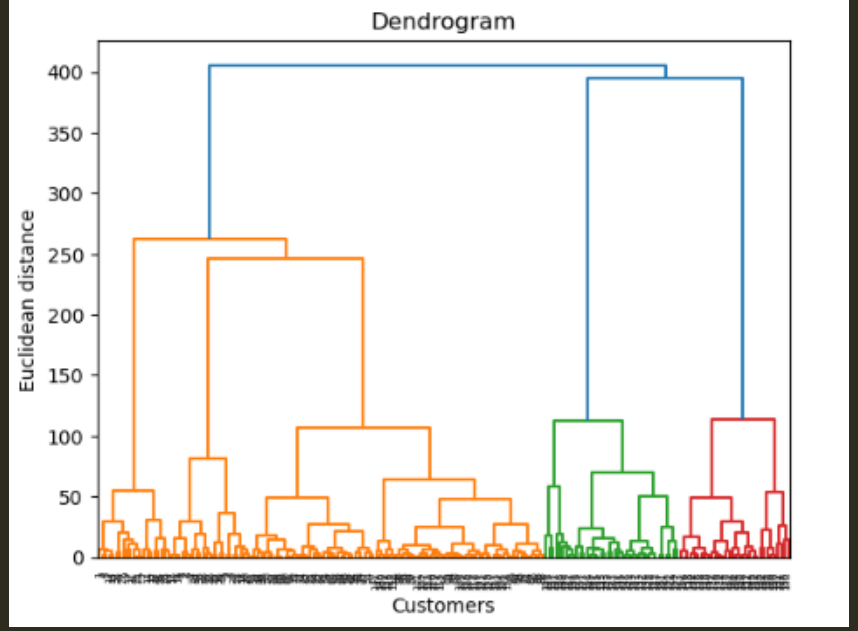
Working Steps:

1. Start with each point as a single cluster
2. Compute pairwise distances between clusters
3. Merge **closest clusters** based on linkage method
4. Repeat steps 2–3 until desired number of clusters is reached

Advantage: No need to pre-specify K, Dendrogram gives visual insight into clustering
Disadvantage: Computationally expensive for large datasets ($O(n^2)$), Must choose **linkage method (measures distance b/w clusters)** carefully
Application: customer segmentation, gene/protein clustering

CODING & REAL WORLD EXAMPLE

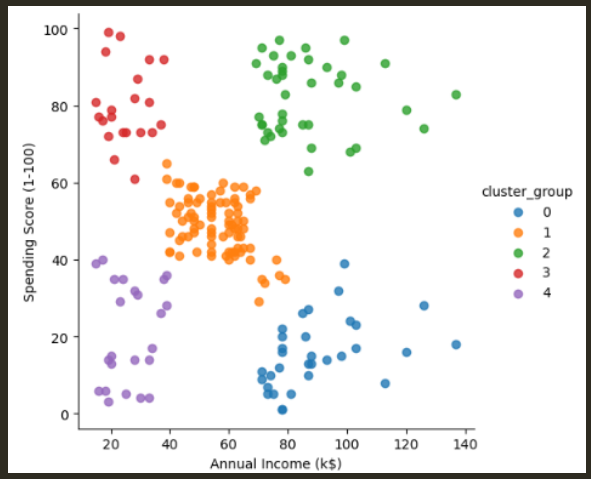
```
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt

dendrogram = sch.dendrogram(sch.linkage(x,method='ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distance')  # this will be calculated by dendrogram
plt.show()

""" 5 big wards in dendrogram  = No of Clusters = 5 (Analyzed from the graph ) """
```



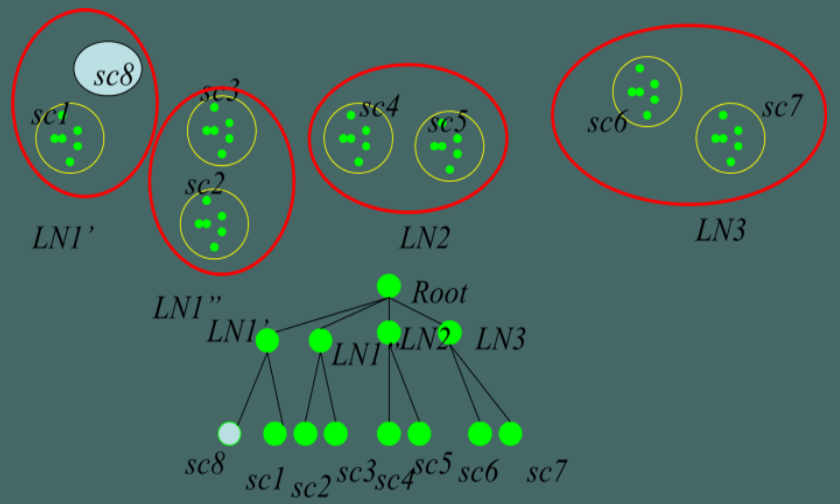
Model Creation

```
from sklearn.cluster import AgglomerativeClustering
agglo_model = AgglomerativeClustering(n_clusters=5)
y_label = agglo_model.fit_predict(x)
y_label
```

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2])
```



BIRCH = CF TREE + THRESHOLD + OPTIONAL K-MEANS (DEFINITION & HOW IT WORKS)



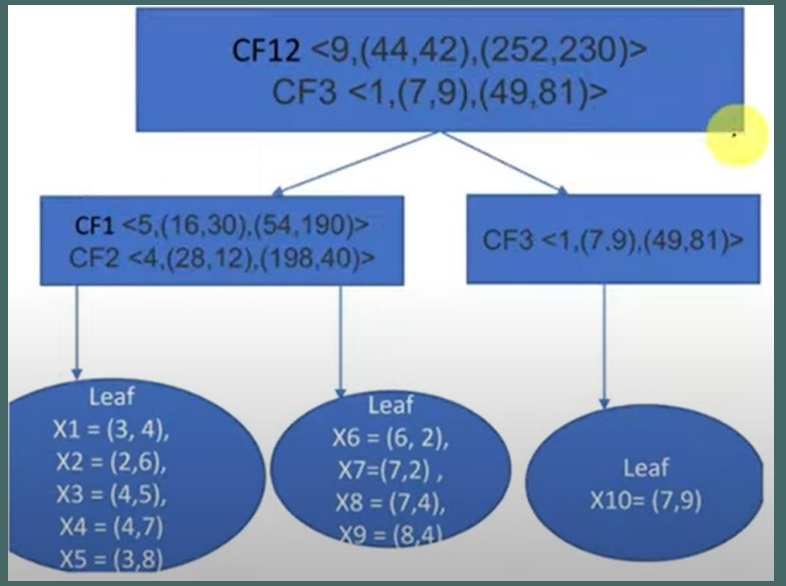
BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a **hierarchical clustering algorithm** designed for **large datasets:**

- Summarizes data using **CF (Clustering Feature) trees**
- Can handle **large-scale datasets efficiently where memory is limited**
- Can be combined with other clustering algorithms (like K-Means) for final refinement

Build a **tree structure (CF Tree)**: each node summarizes a cluster using a **Clustering Feature (CF)** tuple:

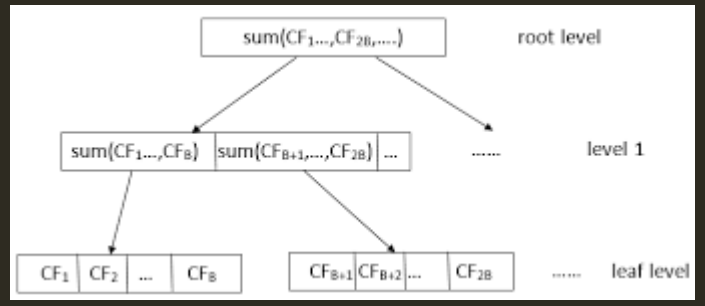
CF=(N,LS,SS)CF = (N, LS, SS)

- **N** = number of points
- **LS** = linear sum of points
- **SS** = square sum of points



Note: Leaf = cluster (each cluster radius must be less than threshold (max radius))

CODING & REAL WORLD EXAMPLE



Working steps (conceptual flow)

1.Build CF Tree

1. Insert points one by one
2. Merge points into leaf nodes if radius < threshold

2.Optional condense or refine

1. Merge small sub-clusters or split large ones

3.Global clustering (optional)

1. Apply K-Means or another algorithm on leaf CF summaries

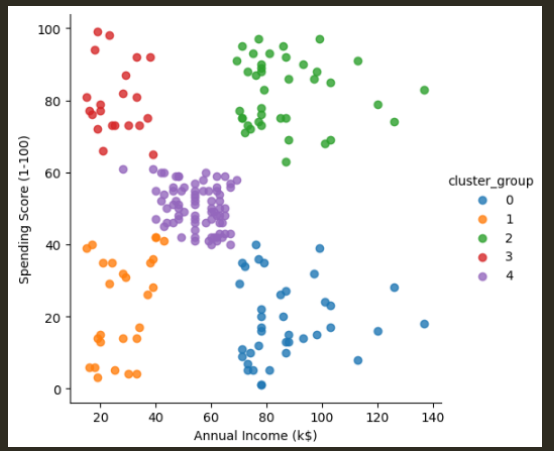
4.Assign labels

1. Leaf clusters → final cluster assignments

```python
# Model creation
from sklearn.cluster import Birch
Birch_model = Birch(threshold=0.4, branching_factor=50, n_clusters=5)
y_label = Birch_model.fit_predict(x)
print(y_label)
```

```
[1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 4 1 3 1 3 1 3 1 3 1
 3 1 3 1 3 1 4 1 3 4 4 1 1 4 4 4 4 4 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2
 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```

Parameter	Meaning
branching_factor	Max number of children per node
threshold	Max radius of sub-clusters stored in leaves



ADVANTAGE & DISADVANTAGE , APPLICATIONS

Advantages of Birch clustering

1. **Handles very large datasets efficiently**
2. **Incremental / online learning possible** (add points without rebuilding)
3. **Memory-efficient** via CF tree
4. Can **combine with other algorithms** (e.g., K-Means on leaf clusters)
5. **Fast,** especially for multi-dimensional datasets

Disadvantages of Birch clustering

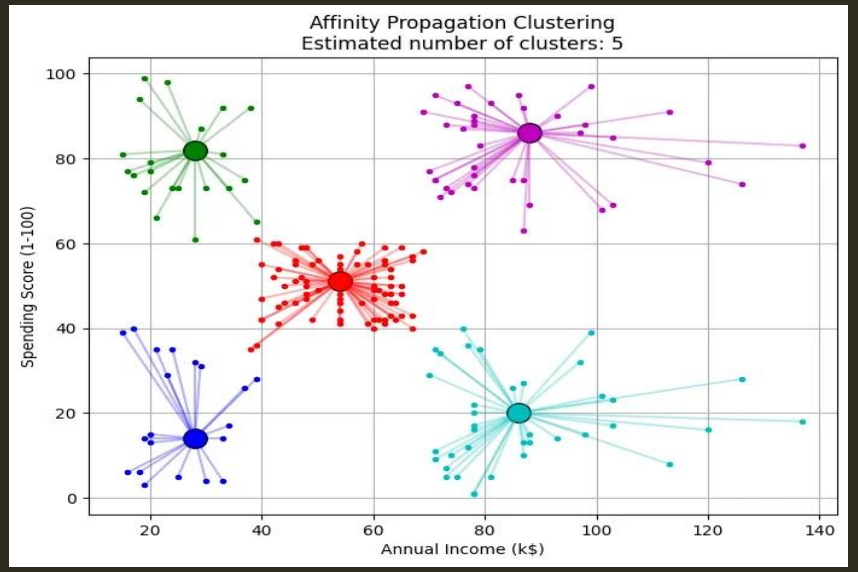
1. Sensitive to **threshold parameter**
2. Hierarchical structure may not always produce **intuitive clusters**
3. Not ideal for **arbitrary-shaped clusters**

Applications of Birch clustering

1. Large-scale **customer segmentation**
2. **Sensor data** clustering for IoT
3. **Document clustering**
4. **Anomaly detection** in large datasets
5. Pre-processing step before **other clustering algorithms**

GRAPH-BASED / MESSAGE PASSING CLUSTERING TYPES

AFFINITY PROPAGATION = SIMILARITY + MESSAGE PASSING + EXEMPLARS (DEFINITION & HOW IT WORKS)



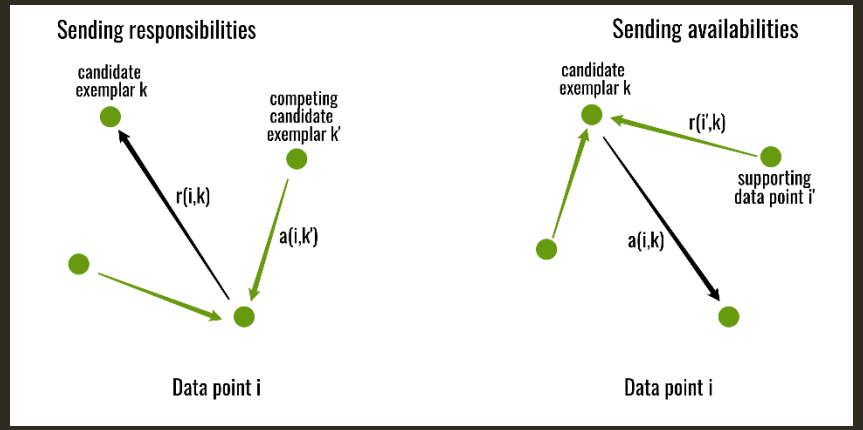
Affinity Propagation is a **message-passing clustering algorithm** that:

- Automatically finds the **number of clusters**
- Selects **real data points as cluster centers** (called *exemplars*)
- Works based on **pairwise similarity between points** until convergence. pairs of samples until convergence.

Each data point sends **two types of messages** to every other point:

- **Responsibility (r)**
  "How suitable is point k to be the exemplar (center) for point i?"
- **Availability (a)**
  "How appropriate is it for point i to choose point k as its exemplar?"

Through iterations: Stable exemplars emerge and Clusters form around them



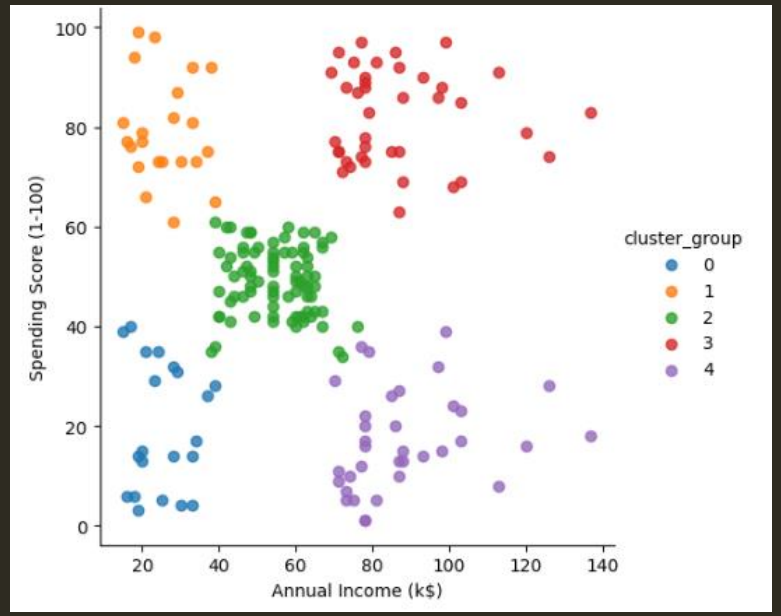
Parameter	Meaning
Preference (very important)	Controls number of clusters
damping	Prevents oscillation
affinity	Similarity measure

CODING & REAL WORLD EXAMPLE

```
# Model creation
"""damping tuning-  range [0.5, 0.9]"""
AffinityPropagation_model = AffinityPropagation(damping=0.5,preference=-30.26622723099282 , affinity='euclidean',random_sta
y_label = AffinityPropagation_model.fit_predict(x)
y_label
```

Working steps (conceptual flow)

1. Compute similarity matrix (usually negative squared Euclidean distance)
2. Initialize responsibility and availability matrices
3. Iteratively update:
    1. Responsibility
    2. Availability
4. Apply damping to stabilize updates
5. Identify exemplars
6. Assign each point to nearest exemplar



ADVANTAGE & DISADVANTAGE , APPLICATIONS

Advantages of Affinity Propagation

1. **No need to specify number of clusters**
2. **Deterministic** (no random initialization like K-Means)
3. **Uses real data points as cluster centers**
4. Works well when **clusters are well separated**

Disadvantages of Affinity Propagation

1. **Computationally expensive ($O(N^2)$)**
   - Not suitable for large datasets
2. **Very sensitive to preference**
3. High memory usage (similarity matrix)
4. Performs poorly on **overlapping or noisy data**

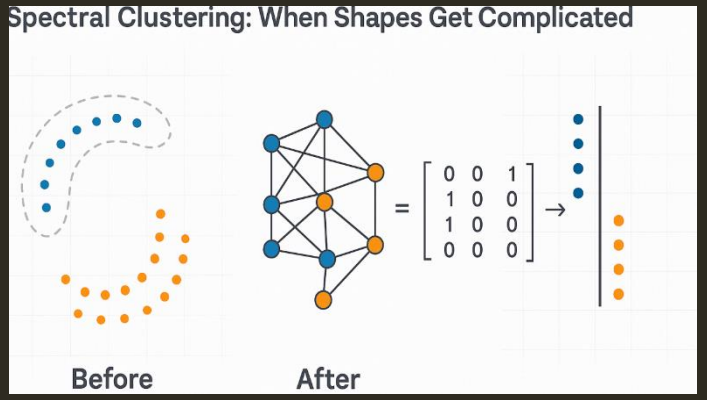
Applications of Affinity Propagation

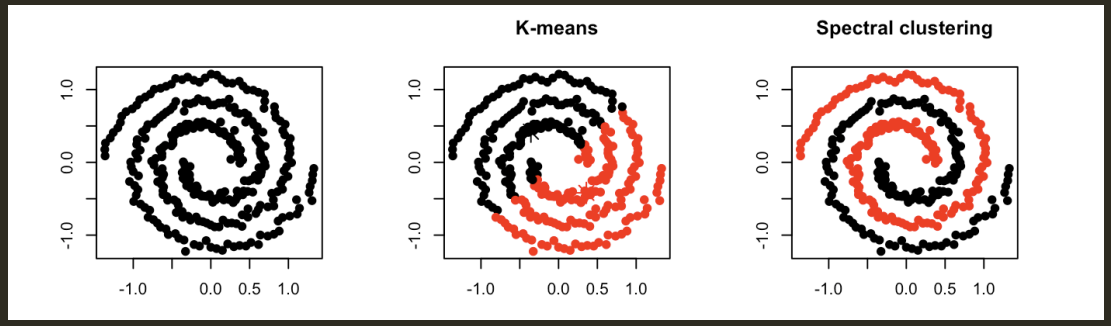
1. **Document / text clustering**
2. **Image segmentation**
3. **Customer segmentation (small datasets)**
4. **Biology (gene/protein clustering)**
5. **Recommendation systems**
6. **Social network analysis**

SPECTRAL CLUSTERING = GRAPH + LAPLACIAN + EIGENVECTORS + K-MEANS (DEFINITION & HOW IT WORKS)

Spectral Clustering is a **graph-based clustering algorithm** that:

- Treats data as a **graph**, where points are nodes and edges represent similarity.
- Uses **eigenvalues/eigenvectors of the Laplacian matrix** to reduce dimensionality.
- Useful for **non-convex clusters** where K-Means fails.



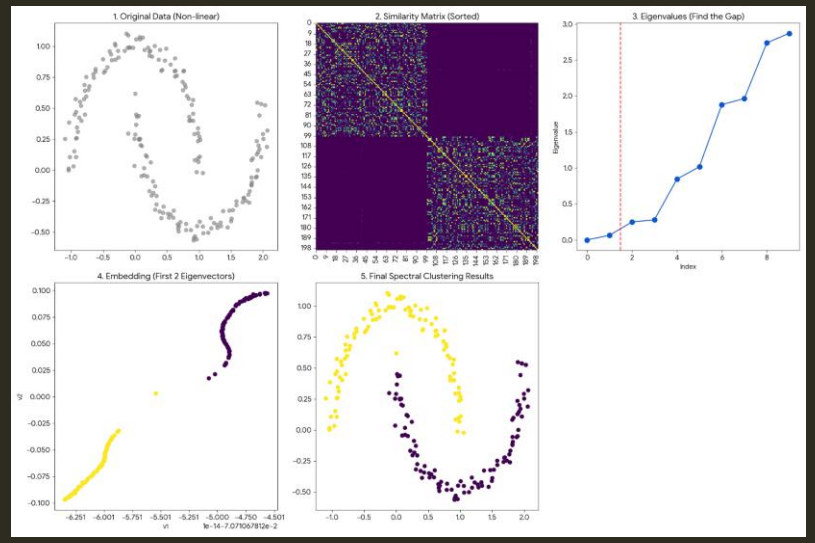


ALGORITHM WORK FLOW:-

1) Build a **similarity matrix** S between points (e.g., Gaussian RBF).
2) Compute the **Laplacian matrix** L=D−S (D-diagonal matrix) or normalized Laplacian.
3) Find the **first k eigenvectors** of LLL → this reduces dimensionality.
4) Apply **K-Means** on the eigenvector representation.
5) Points in the same group of eigenvectors → same cluster.

Main takeaway: Spectral Clustering transforms complex structures into a space where **clusters are linearly separable**.

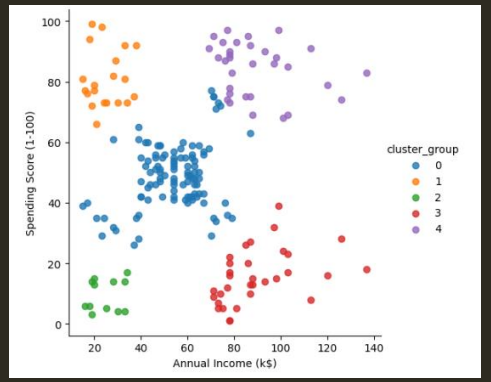
CODING & REAL WORLD EXAMPLE




```
# Model creation
from sklearn.cluster import SpectralClustering
""" based on Domain knowledge, assigned : n_clusters=5 """
SpectralClustering_model = SpectralClustering(n_clusters=5,gamma=1.0, affinity='rbf',n_neighbors=10,assign_labels='kmeans',  random_state=0)
y_label = SpectralClustering_model.fit_predict(x)
y_label
```

```
array([0, 1, 2, 1, 0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 0, 1, 0, 1, 0, 1,
       2, 1, 2, 1, 0, 0, 0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 4, 3, 0, 3, 0,
       0, 0, 3, 4, 3, 0, 3, 0, 3, 4, 0, 4, 3, 4, 0, 4, 3, 4, 3, 4, 3, 4,
       3, 4, 3, 4, 3, 4, 0, 4, 3, 4, 3, 4, 3, 4, 3, 0, 3, 4, 3, 4, 3, 4,
       3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4,
       3, 4])
```

Parameter	Meaning
n_clusters	Number of clusters (must specify)
affinity	How similarity between points is measured (rbf, nearest_neighbors, precomputed)
gamma	Kernel coefficient for RBF kernel
n_neighbors	For nearest neighbors affinity
assign_labels	How to assign clusters in eigenvector space (kmeans by default)



ADVANTAGE & DISADVANTAGE , APPLICATIONS

Advantages of spectral

1. Works well for **non-convex clusters**.
2. Can capture **complex cluster structures** (e.g., circles, moons).
3. Uses **graph-based similarity**, flexible.
4. Can incorporate **kernel tricks** for more complex similarity measures.

Disadvantages of spectral

1. **Must pre-specify number of clusters (k)**.
2. **Computationally expensive** for large datasets (affinity matrix = $O(N^2)$).
3. Sensitive to **affinity/kernel choice** and parameters (gamma, n_neighbors).

Applications of spectral

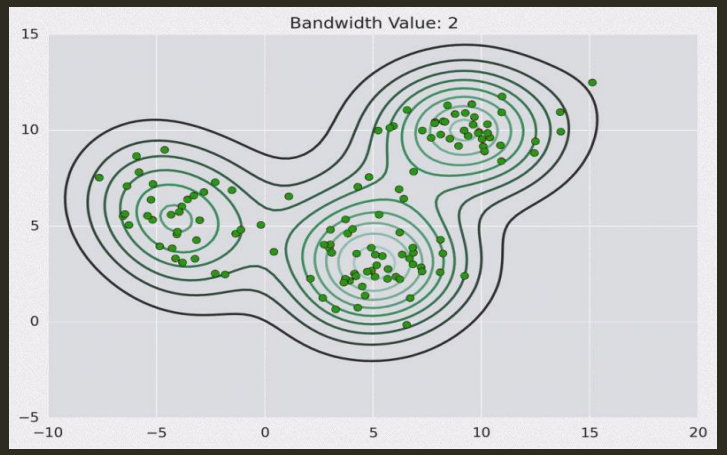
1. Image segmentation (pixels or superpixels)
2. Social network community detection
3. Non-linear clustering (moons, spirals)
4. Document clustering (text similarity graphs)
5. Gene/protein clustering in biology

DENSITY-BASED / MODE-SEEKING CLUSTERING TYPES

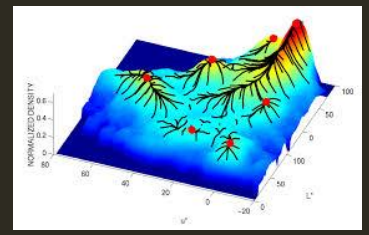
MEAN SHIFT CLUSTERING = DENSITY PEAKS + SLIDING WINDOW + NO K
(DEFINITION & HOW IT WORKS)

Mean Shift is a **density-based clustering algorithm** that:

- Finds **modes (peaks)** of a data distribution
- Shifts points iteratively toward **high-density regions**
- Automatically determines the **number of clusters**



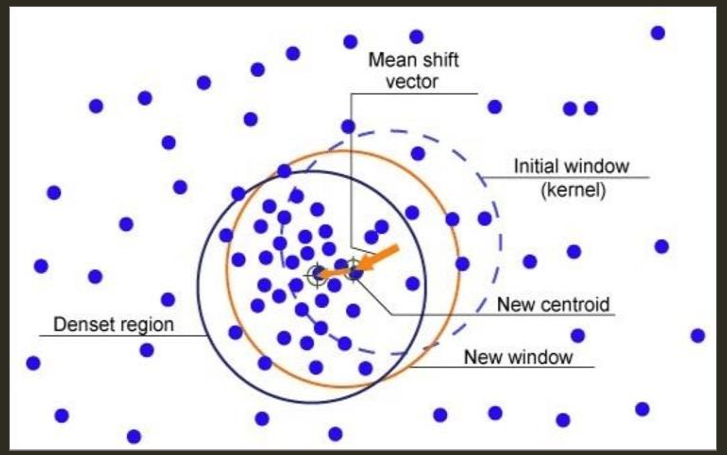
- Hills = dense regions
- Valleys = sparse regions



Mean Shift:

1. Places a **window (kernel)** around a point
2. Computes the **mean of points inside the window**
3. Shifts the window to the mean
4. Repeats until convergence at a **density peak**

Points that converge to the **same peak → same cluster**



Working steps (conceptual flow)

1. Choose a bandwidth
2. For each point:
   1. Find neighbors within bandwidth
   2. Compute mean of neighbors
   3. Shift point to mean
3. Repeat until shift < threshold
4. Merge points converging to same mode
5. Each mode = cluster center

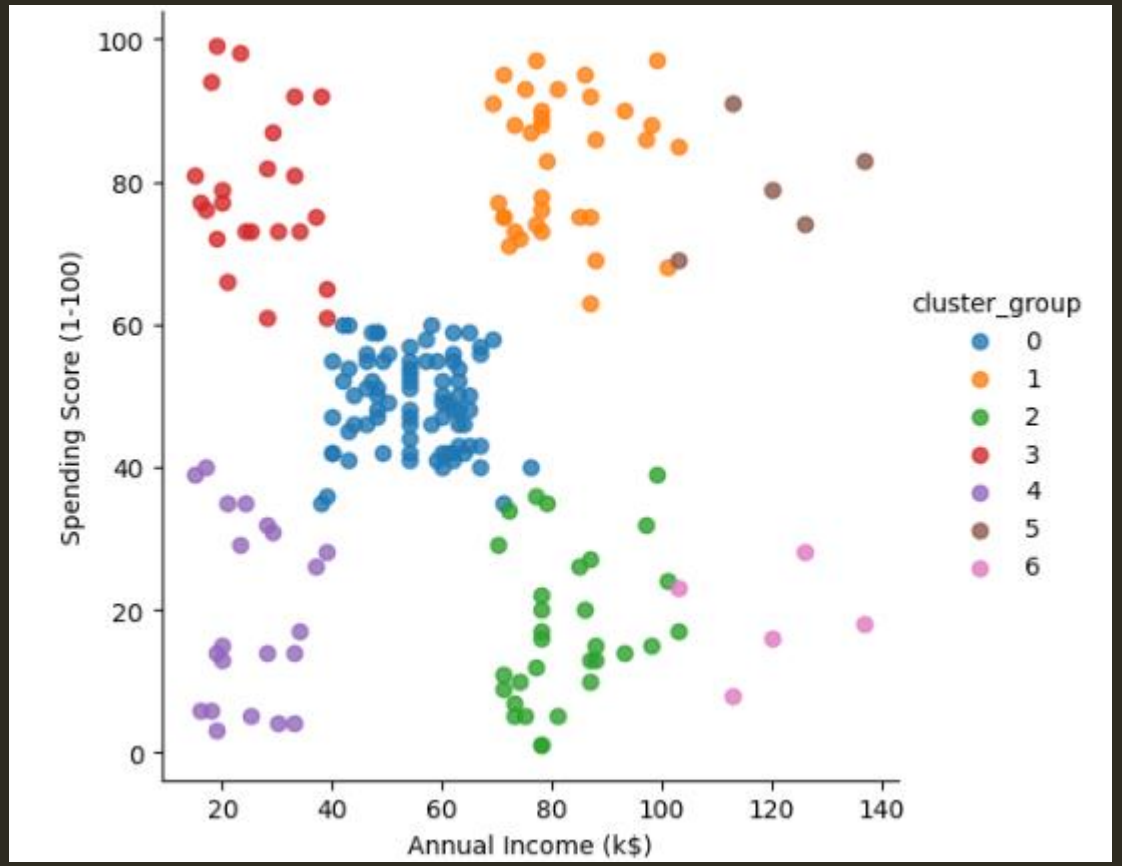
CODING & REAL WORLD EXAMPLE

```
## ESTIMATE THE BANDWIDTH
# https://scikit-learn.org/stable/modules/generated/sklearn.cluster.estimate_bandwidth.html

from sklearn.cluster import estimate_bandwidth
bandwidth = estimate_bandwidth(x, quantile=0.1, n_samples=None)
print("Estimated bandwidth:", bandwidth)
```

```
Estimated bandwidth: 0.6487582521320147
```

```
# Model creation
from sklearn.cluster import MeanShift
MeanShift_model = MeanShift(bandwidth=0.6487582521320147, bin_seeding=True)
y_label = MeanShift_model.fit_predict(x)
y_label
```

Parameter	Meaning
bandwidth	Radius of the window (MOST IMPORTANT)
kernel	Shape of window (Gaussian by default)
bin_seeding	Speed optimization
cluster_all	Assign all points as cluster (true) or all points + mark noise (false)



ADVANTAGE & DISADVANTAGE , APPLICATIONS

Advantages of Mean shift clustering

1. **No need to specify number of clusters**
2. Works well for **arbitrary cluster shapes**
3. Finds **natural density peaks**
4. Deterministic (no random init)

Disadvantages of Mean shift clustering

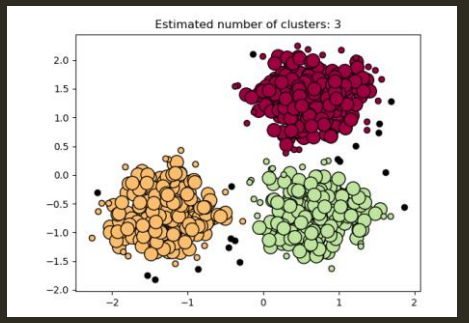
1. **Very slow on large datasets**
2. **Highly sensitive to bandwidth**
3. Struggles with **varying densities**
4. High memory & computation cost

Applications of Mean shift clustering

1. Image segmentation
2. Object tracking (computer vision)
3. Feature space clustering
4. Pattern recognition
5. Small-scale customer segmentation

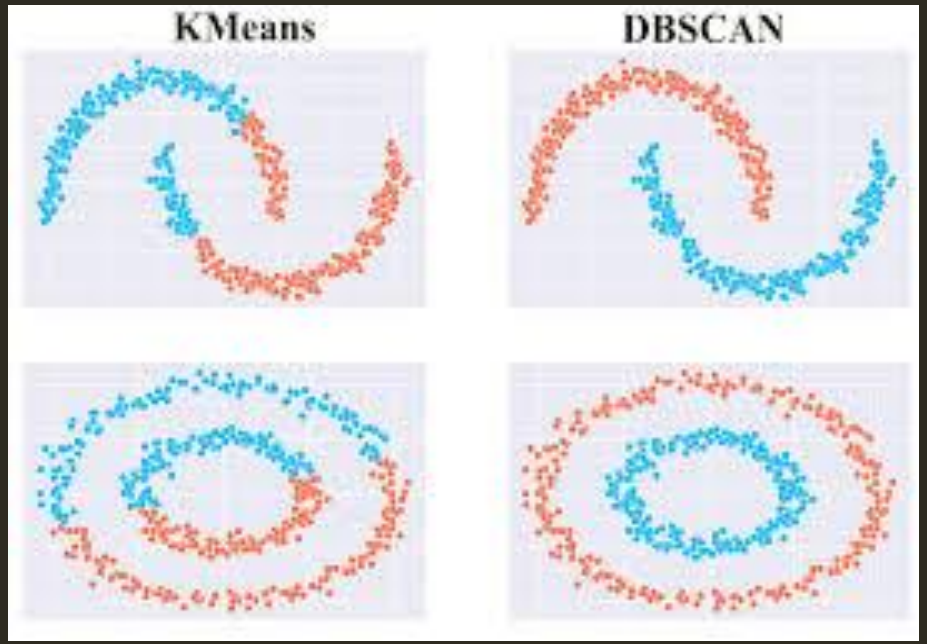
DENSITY-BASED / HIERARCHICAL CLUSTERING TYPES

DBSCAN (DEFINITION & HOW IT WORKS)



DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised machine learning algorithm that groups together closely packed data points into dense regions, identifying clusters of arbitrary shapes, while marking sparse points as noise or outliers

- **Unsupervised, distance-based clustering algorithm**
- Forms clusters based on **data density**
- **Automatically detects outliers (noise)**
- Does **not require the number of clusters in advance**



Workflow

1. **Scale the data** (StandardScaler)
2. **Initialize parameters**: min_samples and eps
3. **Assign point types** (core, border, noise)
4. **Label clusters**

Key Parameters

Min Samples

•Minimum number of neighbouring points required to form a cluster

•Common heuristic:

min_samples = 2 × number of features

(recommended by German researchers; works well in many cases)

•Example:

Features = 2 → min_samples = 4

Epsilon (eps)

•Radius within which neighbours are searched

•Determines cluster density

•Selected using the **k-distance graph (elbow method)**

Point Types

•**Core points** → dense central points

•**Border points** → edge points (still part of a cluster)

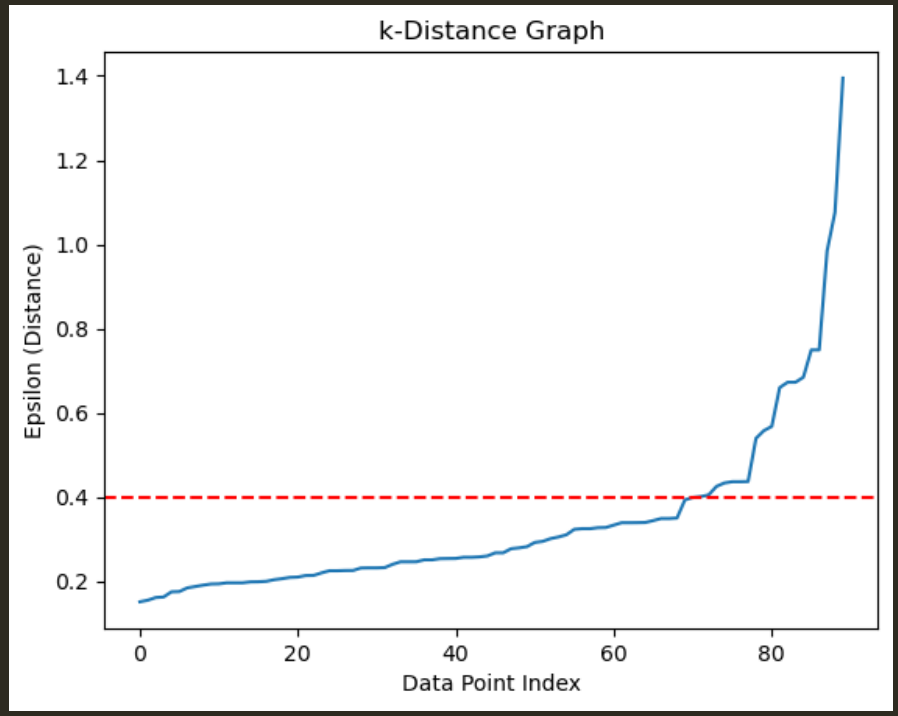
•**Noise points** → outliers (not part of any cluster)

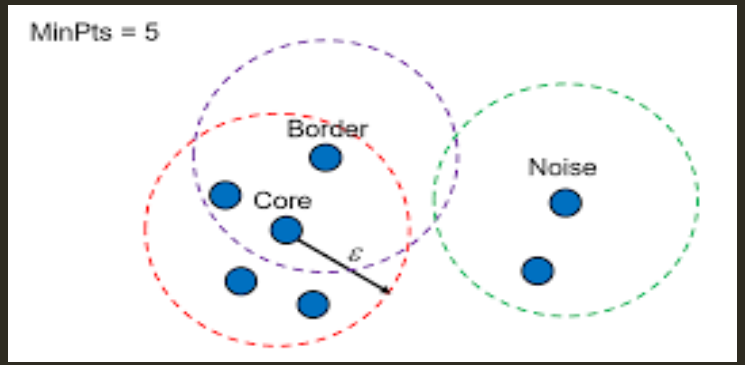
Cluster Labelling

•Each cluster contains **both core and border points**

•Noise points are labelled as **-1**

•Same process is repeated to identify multiple clusters



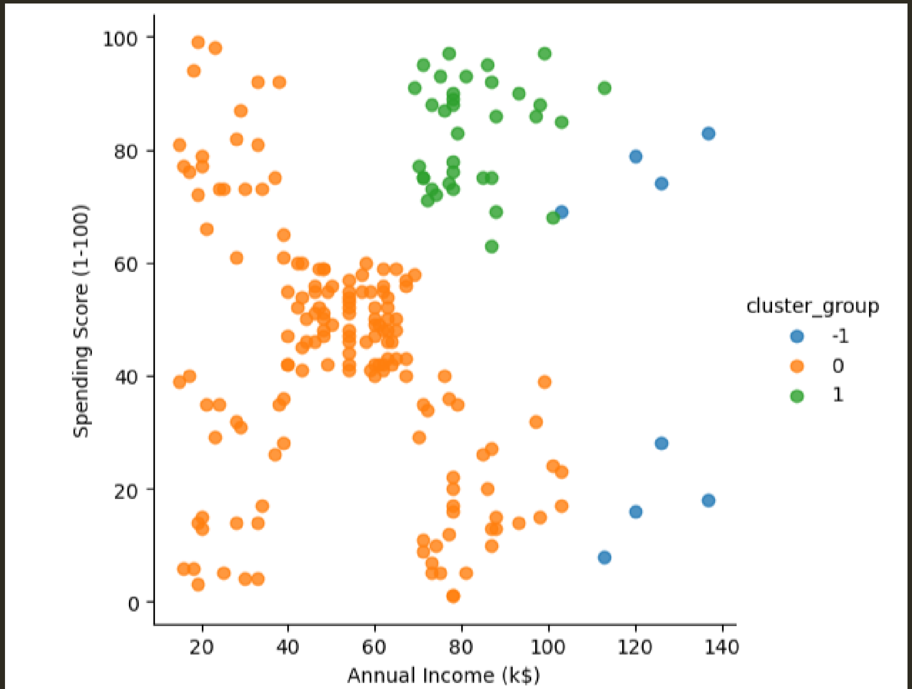


CODING & REAL WORLD EXAMPLE

Model Creation

```python
from sklearn.cluster import DBSCAN
DBSCAN = DBSCAN(eps=0.5, min_samples=MinSamples)
y_label = DBSCAN.fit_predict(x)
y_label
```

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



Parameter	Meaning
eps	Max distance for points to be neighbors
min_samples	Minimum points to form a dense region (core point)

ADVANTAGE & DISADVANTAGE , APPLICATIONS

Advantages of DBSCAN

- Does **not require predefined number of clusters**
- Can find **arbitrarily shaped (non-spherical) clusters**
- **Effectively detects outliers / noise**
- Works well when clusters are **clearly separated by density**
- Robust to small variations in data

Disadvantages of DBSCAN

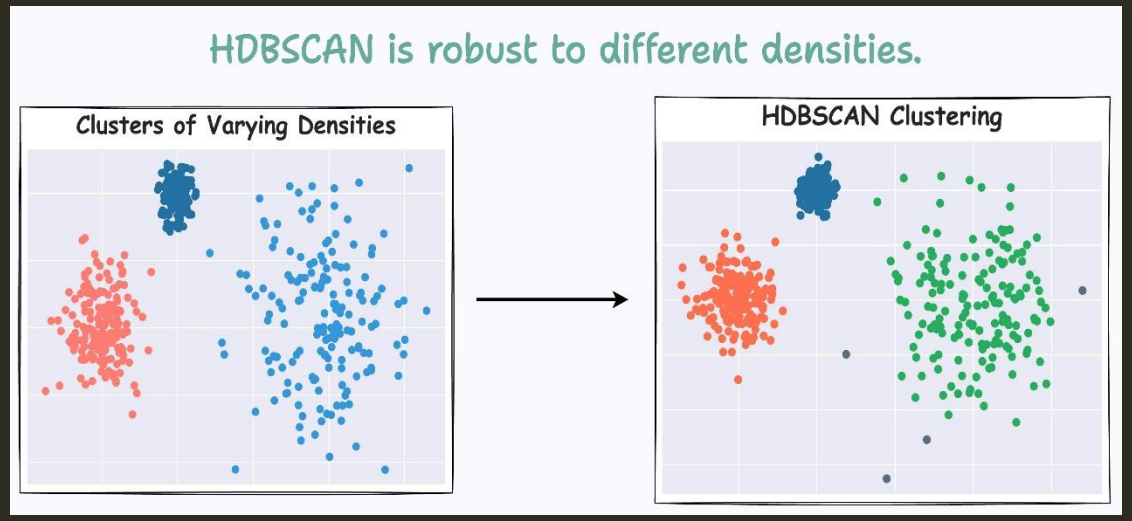
- Difficult to choose optimal **eps** and **min_samples**
- Performs poorly when data has **varying densities**
- Sensitive to **distance metric** and **data scaling**
- Struggles with **high-dimensional data** (curse of dimensionality)
- Not ideal for very large datasets without optimization

Applications of DBSCAN

 **Business & Analytics -** Customer segmentation
 **Geospatial & Location Data** - Location-based service analysis
 **Anomaly & Fraud Detection** - Credit card fraud detection
 **Scientific & Industrial** - Pattern detection in medical data, Manufacturing defect analysis

HDBSCAN (DEFINITION & HOW IT WORKS)

HDBSCAN is an advanced version of DBSCAN that automatically finds clusters of different densities and separates noise, without needing to manually choose eps.



Key Features

- **Varying Density Clusters**
- **Automatic Cluster Number**
- **Noise Handling**
- Creates a hierarchy of clusters
- **Parameter Intuition** - `min_cluster_size`

HDBSCAN does not fix one epsilon like DBSCAN.
Instead, HDBSCAN builds a hierarchy by varying the density threshold instead of fixing eps. It observes how clusters form and dissolve as eps increases and selects clusters that are stable over a wide range of density values. This allows it to handle clusters with varying densities automatically

HDBSCAN Workflow

1) Scaling Dataset
2) Initialize min_cluster_size - (according to dataset size) – tune it .. Hdbscan handle min_sample automatically
3) Find Core Distance (distance for each point to kth nearest neighbour)
4) Find MRD – Mutual Reachability Distance
5) Construct the **minimum spanning tree**

Max (core distance(A), core distance(b), Euclidean distance(A,B)) = MRD (A,B)

MRD is less – A B belongs to same cluster

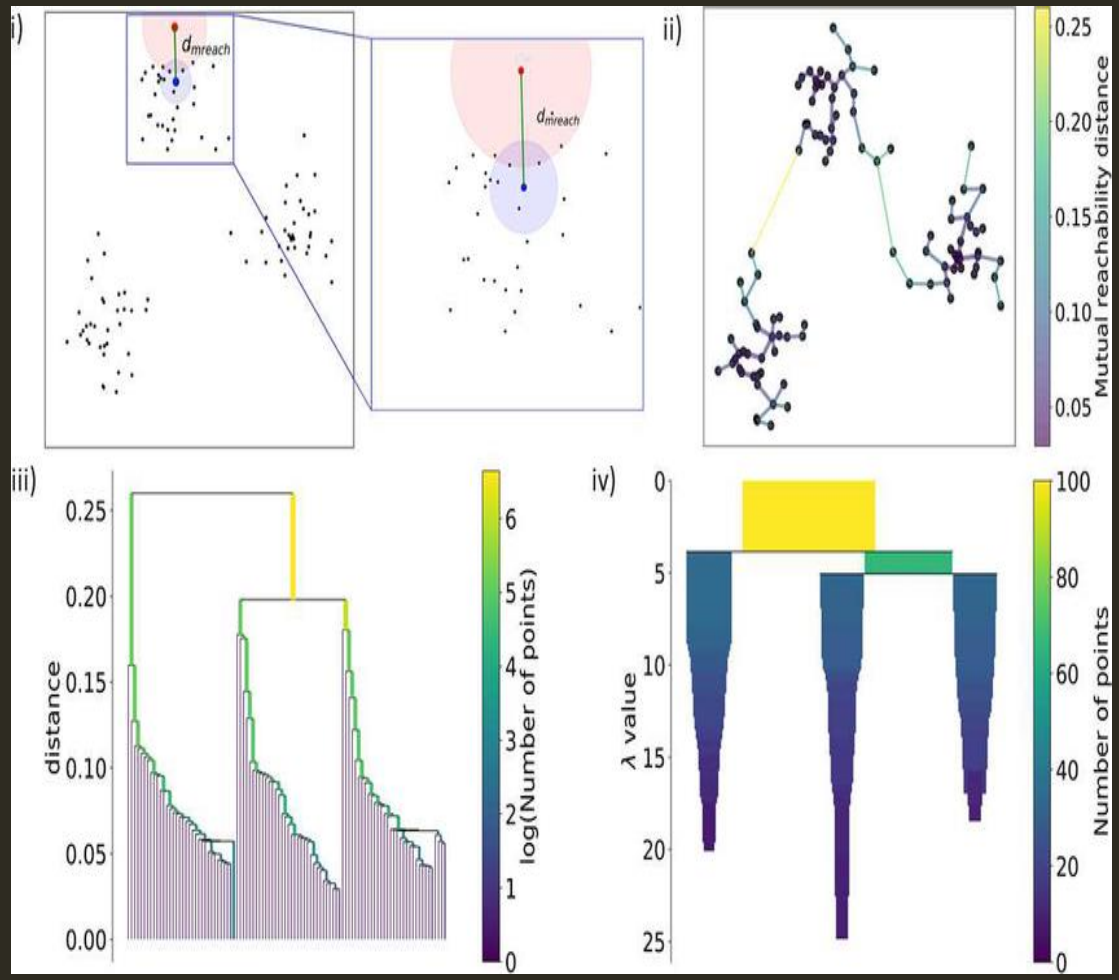
MRD is BIG – A B belongs to different cluster

6) Adjust the density threshold to find the clusters (by automatic tuning of eps 0 – max in the minimum spanning tree -> done by the algorithm)

7) Build condensed tree

calculate the stability score

Stability = sum of densities * number of points over its lifetime in thresholds

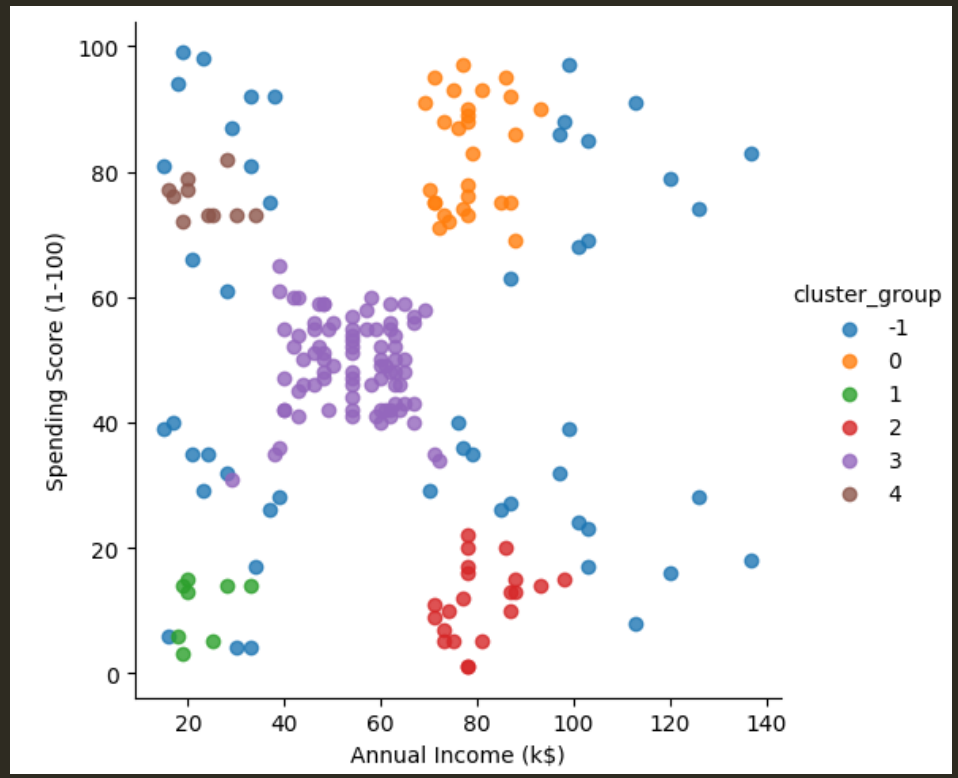


CODING & REAL WORLD EXAMPLE

Model Creation

In [74]:
```
from sklearn.cluster import HDBSCAN
hdbscan = HDBSCAN(min_cluster_size=8, min_samples=None )   # tune 5-10 => min_cluster_size
y_label = hdbscan.fit_predict(x)
y_label
```

Out[74]:
```
array([-1, -1, -1,  4, -1,  4,  1, -1,  1,  4,  1, -1,  1,  4,  1,  4, -1,
       -1, -1, -1, -1,  4,  1,  4,  1,  4, -1, -1,  3, -1, -1,  4, -1, -1,
        1, -1, -1,  4, -1, -1,  3, -1,  3,  3, -1,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  0, -1,  0,  3,  0,  2,  0,  2,  0,  3,  0,  2,  0,
        2,  0,  2,  0,  2,  0, -1,  0,  2,  0, -1,  0,  2,  0,  2,  0,  2,
        0,  2,  0,  2,  0,  2,  0, -1,  0,  2,  0, -1,  0,  2,  0, -1, -1,
        2,  0,  2,  0,  2,  0,  2,  0,  2,  0, -1, -1,  2, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1], dtype=int64)
```



Parameter	Meaning
min_cluster_size	Minimum size of a cluster (most important)
min_samples	Minimum points to define core point (optional)

ADVANTAGE & DISADVANTAGE , APPLICATIONS

Advantages of HDBSCAN

- Finds clusters of **varying densities**
- Detects **noise/outliers automatically**
- **No need to specify number of clusters**
- Handles **arbitrary-shaped clusters**
- Produces **stable clusters** via condensed tree

Disadvantages of HDBSCAN

- **Slower** than DBSCAN/K-Means on large datasets
- **Requires min_cluster_size tuning**
- **Harder to interpret** (hierarchy + stability scores)
- Sensitive to **high-dimensional data**

Applications of HDBSCAN

- Customer segmentation
- Anomaly/outlier detection
- Geospatial clustering (hotspots)
- Biology/genomics (genes, cells)
- Image analysis & computer vision
- Market basket / e-commerce patterns

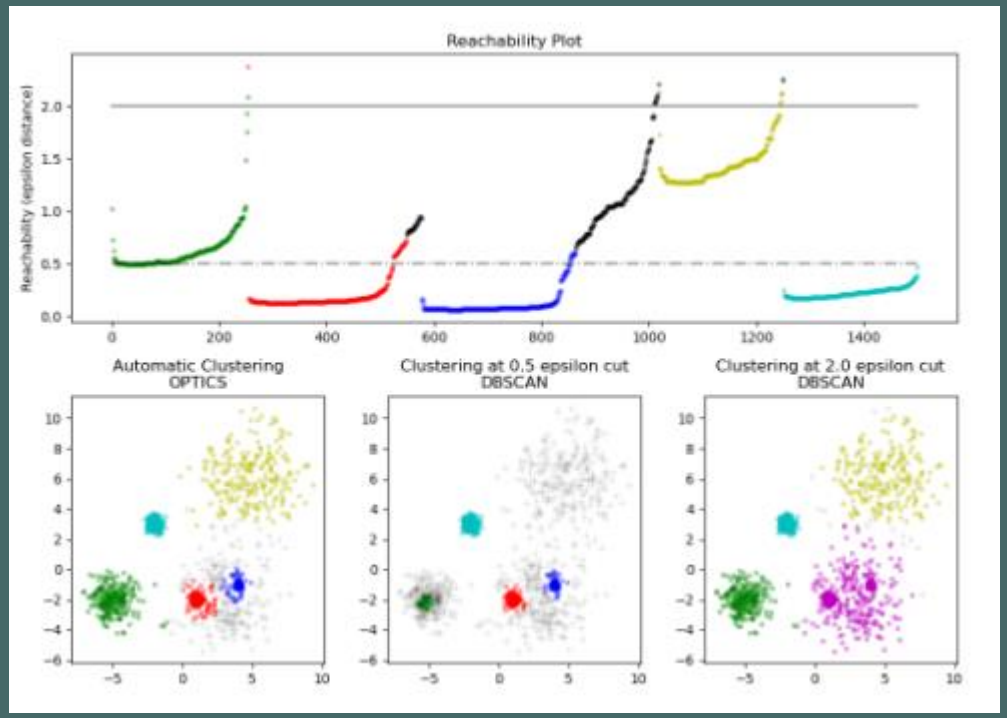
OPTICS (DEFINITION & HOW IT WORKS)

OPTICS = **Ordering Points To Identify the Clustering Structure**
It is a **density-based clustering algorithm**, similar to **DBSCAN**, but more flexible.

- Unlike DBSCAN, it **doesn't require a single eps (radius) to define clusters**.
- OPTICS creates a **reachability plot** showing density-based cluster structure at multiple scales.
- It can detect clusters with **varying densities** in a dataset.

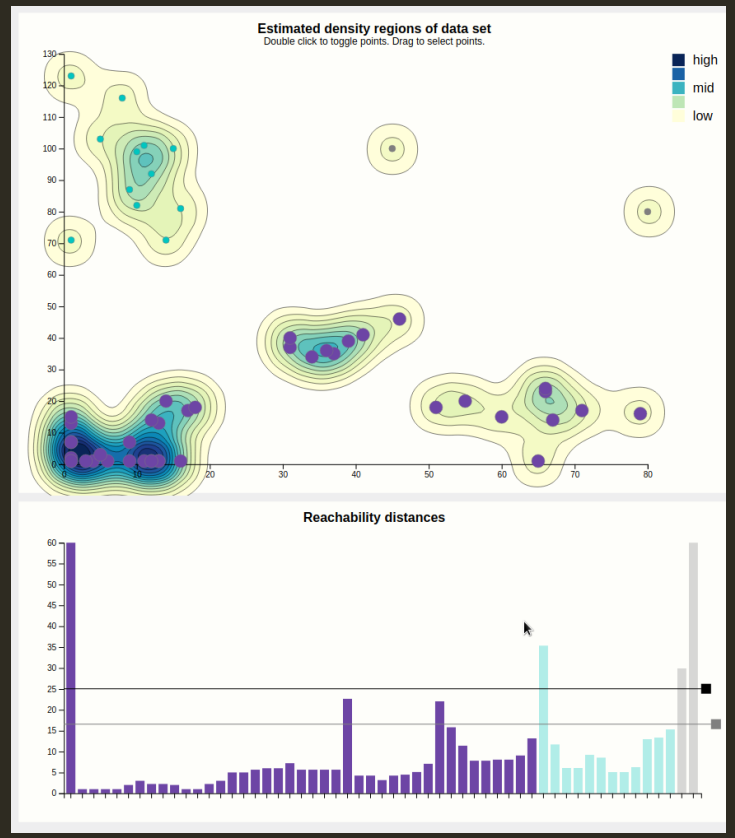
Key Idea:

- Points are ordered by **density-reachability**.
- Dense regions → clusters. Sparse regions → noise.
- It handles noise better than K-Means and Hierarchical clustering.



Algorithm Steps:

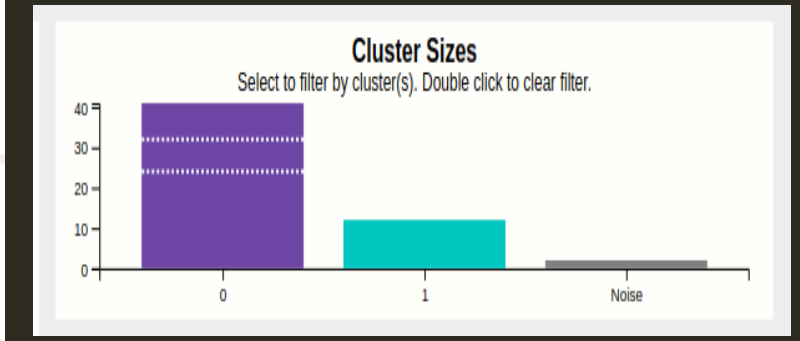
- Start with an unvisited point.
- Compute its **core distance.**
- Expand to neighbors in order of **reachability distance.**
- Continue until all points are processed → gives a **reachability plot.**
- Use a threshold on reachability distances to extract clusters. (Xi cut)



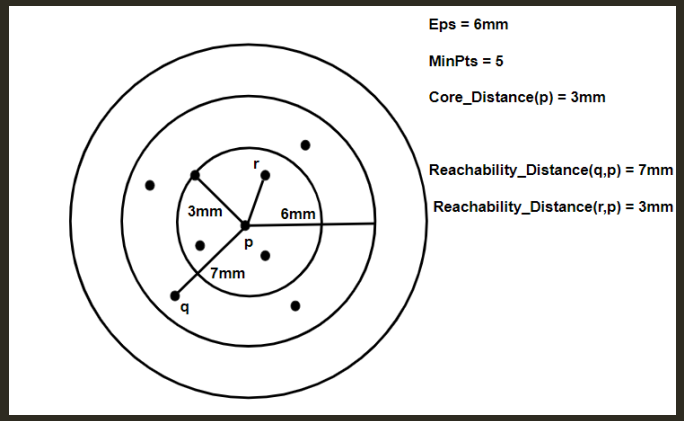
Reachability distance asks:

RD measures **density-aware closeness**

"How difficult is it to reach point p starting from a dense region (o)?"



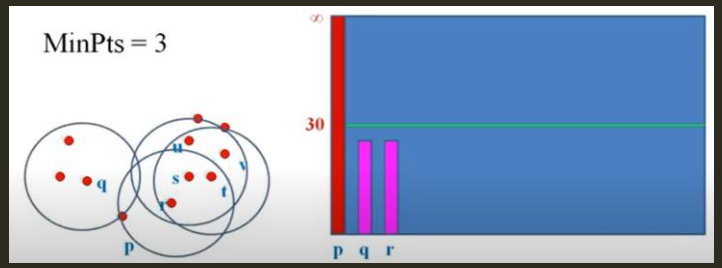
Reachability Distance

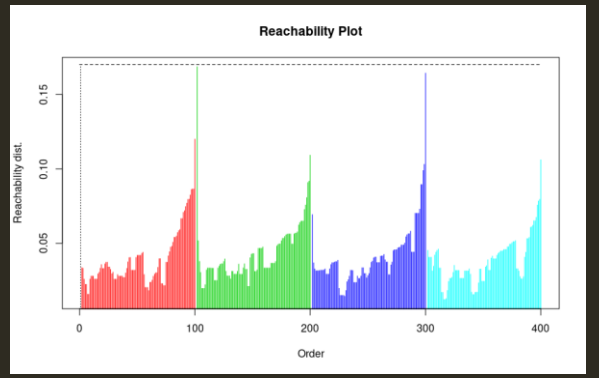


o → core point

$$RD(p,o) = \max(CD(o), \text{distance}(p,o))$$

Reachability plot.

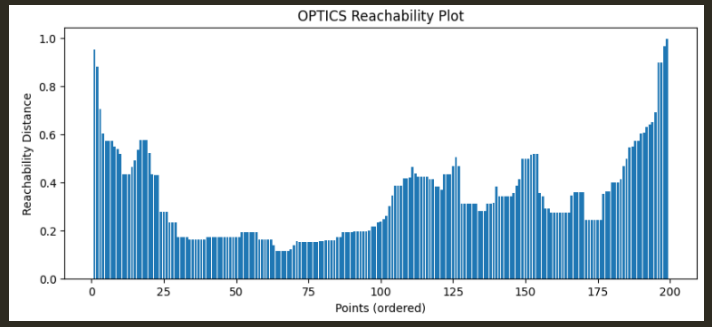


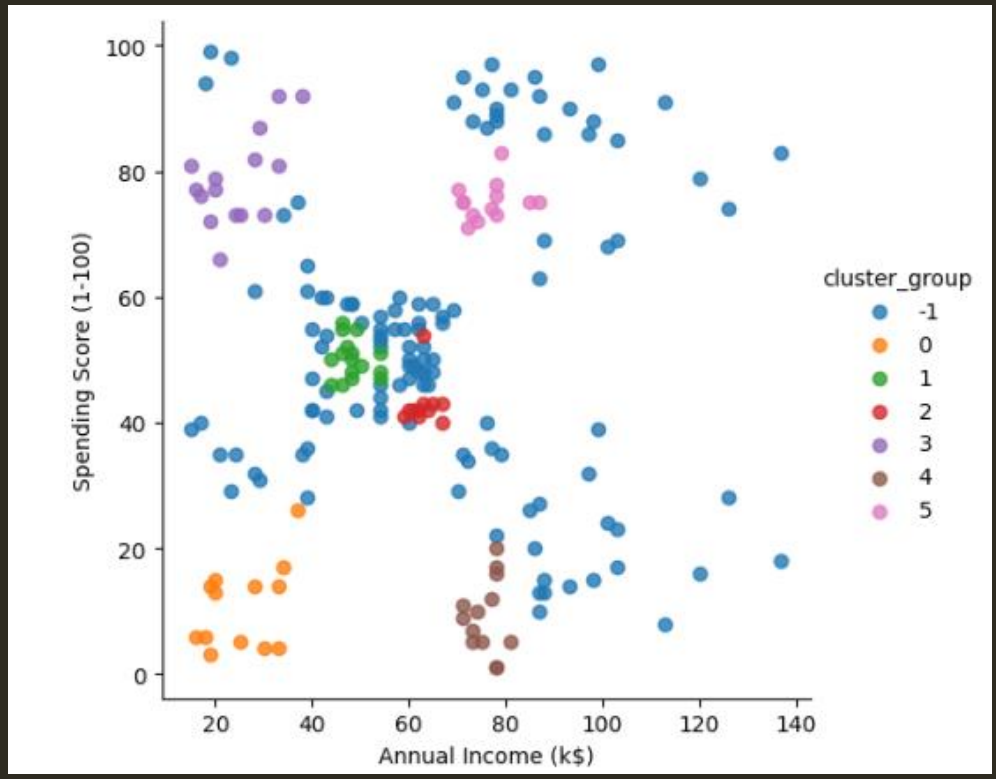


- **Valleys** = dense clusters.
- **Peaks** = sparse/noise regions.

CODING & REAL WORLD EXAMPLE

```
from sklearn.cluster import OPTICS
"""
Xi - tuning range:-
0.01 → 0.2 is common.
Smaller values (e.g., 0.01) → More fine-grained clusters.
Larger values (e.g., 0.1-0.2) → Coarser clusters, fewer splits.
"""
optics_model = OPTICS(min_samples=10,xi=0.01,min_cluster_size=10)
y_label = optics_model.fit_predict(x)
y_label
```





Explanation of Parameters:

- min_samples: Like DBSCAN → minimum points to form a cluster (density).
- xi: Determines steepness threshold to identify clusters in reachability plot. (cluster boundary)
- min_cluster_size: Minimum fraction or number of points to form a cluster.

ADVANTAGE & DISADVANTAGE , APPLICATIONS

Advantages of OPTICS

- Can detect **clusters of varying density** (unlike DBSCAN).
- Automatically handles **noise** points.
- Produces **hierarchical cluster ordering** → more insights.
- No need to predefine a single radius (eps).

Disadvantages of OPTICS

- Computationally more **expensive** than DBSCAN and K-Means (O(n log n) to $O(n^2)$).
- Harder to interpret reachability plots for very large datasets.
- Sensitive to min_samples and xi parameters.

Applications of OPTICS

- **Geospatial clustering**: finding regions of high activity (crime, disease, etc.)
- **Astronomy**: clustering stars or galaxies with varying densities
- **Marketing**: customer segmentation with variable purchase patterns
- **Anomaly detection**: sparse points can be considered anomalies
- **Image processing**: detecting dense object areas

CLUSTERING EVALUATION METRICS

Silhouette Score

$$\text{Silhouette} = \frac{b - a}{\max(a, b)}$$

Silhouette Score tells you **how well each data point fits into its assigned cluster.**

It checks **two things for every point:**

1. Intra cohesion = Intra-cluster distance
2. Inter separation = Nearest-cluster distance

Value	Meaning
+1 (close to 1)	Excellent clustering
~0	Overlapping clusters
< 0	Point is probably in the wrong cluster

```
from sklearn.metrics import silhouette_score
score = silhouette_score(x, y_mean_predicted)
print("Silhouette Score:", score)
```

Davies–Bouldin Index (DBI)

Measures **average similarity between each cluster and its most similar cluster**

Lower value = better clustering (Range: 0 to ∞)

⇒ good for DBSCAN/HDBSCAN

```
from sklearn.metrics import davies_bouldin_score
dbi = davies_bouldin_score(x, y_label)
print("Davies-Bouldin Index:", dbi)
```

Density-Based Clustering Validation

- DBCV (BEST for DBSCAN / HDBSCAN)
- Designed **specifically for density-based clustering**
- Uses **density separation,** not distance
- **Available in 'hdbscan' library**

Higher = better **(Range -1 to 1)**

```
import hdbscan
from hdbscan.validity import validity_index
dbcv = validity_index(x, y_label)
print("DBCV Score:", dbcv)
```

THANK YOU!

Happy Learning!!