Machine Learning with Python(M1) (1)

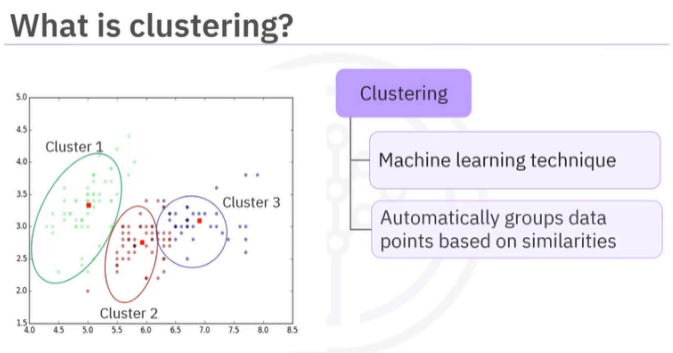
Created	@December 8, 2025 11:32 PM
Module Code	IBMM01: Introduction to Machine Learning

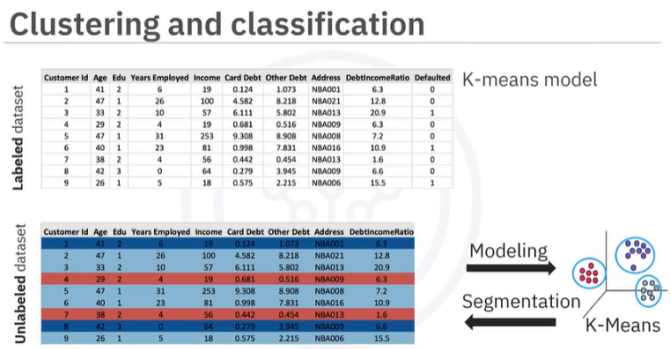
Module 4: Building Unsupervised Learning Model

Clustering

1. What is clustering?

- **Unsupervised learning** method.
- Groups data points into clusters based on **similarity**, without using labels.
- Used when labels are NOT available (unlike classification).
- can use a single feature or multiple features
- classification technique but works with unlabeled data



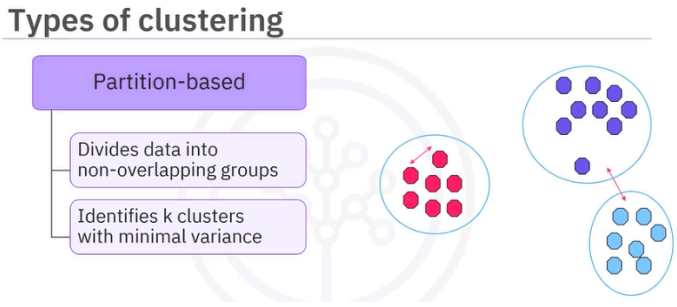


2. Applications of Clustering

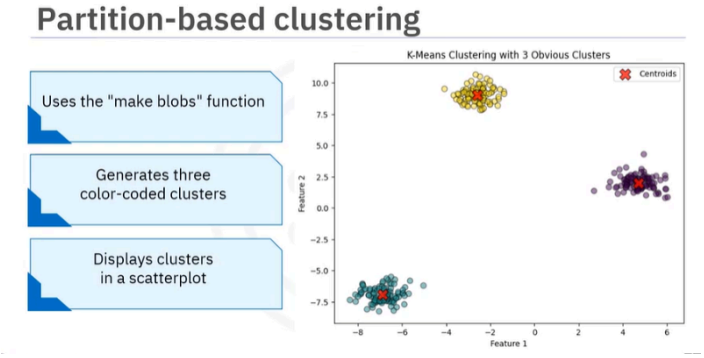
- **Customer segmentation** (marketing, e-commerce)
- **Music genre grouping**
- **User behavior analysis**
- **Image segmentation** (medical anomaly detection, object separation)
- **Anomaly detection: detecting outliers** (fraud, machine faults)
- **Feature engineering** (create new features, reduce dimensionality, improves model performance)
- **Data summarization** (e.g., image compression → cluster centers reduce size)
- pattern recognition: groups objects and aids in image segmentation

3. Types of Clustering

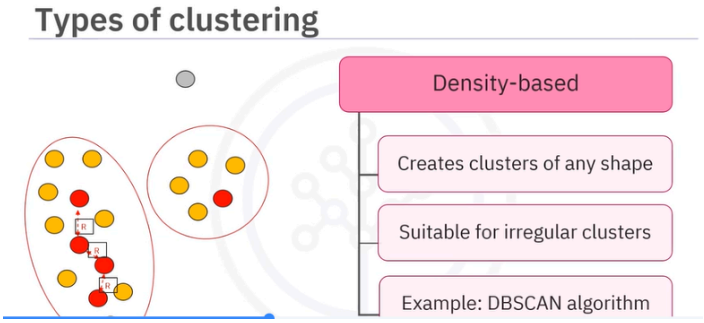
A. Partition-based



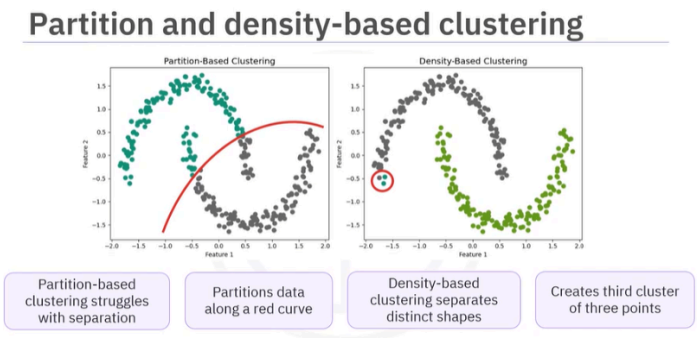
- Example: **K-means**
- Divides data into **k non-overlapping clusters**
- Works best with **spherical clusters**
- Fast & scalable
- Struggles with **non-linear shapes** (e.g., moon shapes)



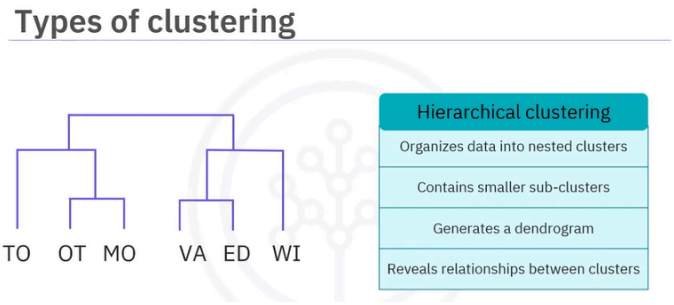
B. Density-based



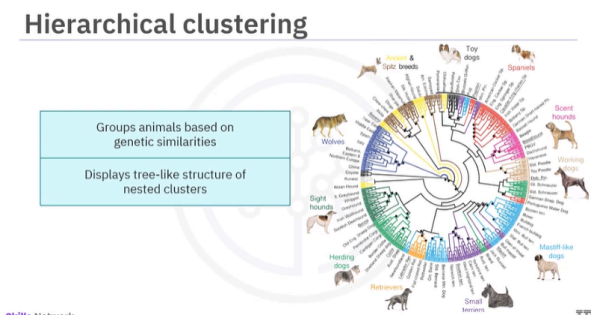
- Example: **DBSCAN**
- Forms clusters of any shape
- Detects noise/outliers
- Useful for **irregular shapes**
- May create extra small clusters if density varies



C. Hierarchical Clustering

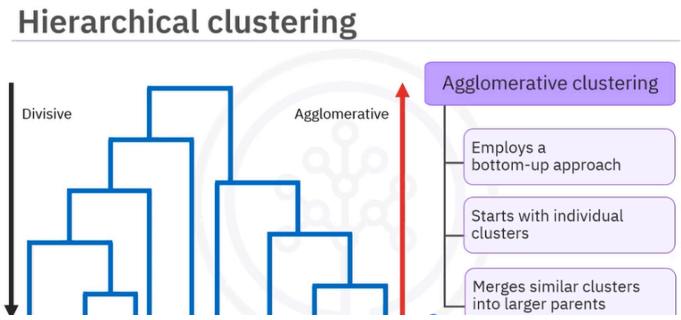


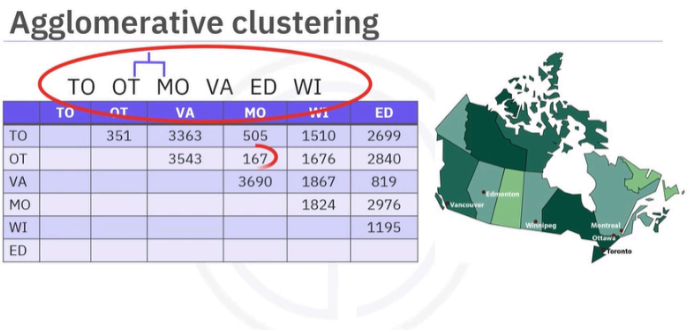
- Builds a **tree of clusters (dendrogram)**
- Two strategies:
  - **Agglomerative (bottom-up)**
    - Start: each data point is its own cluster
    - Merge closest clusters until one big cluster
  - **Divisive (top-down)**
    - Start: all points in one cluster
    - Split into smaller clusters repeatedly

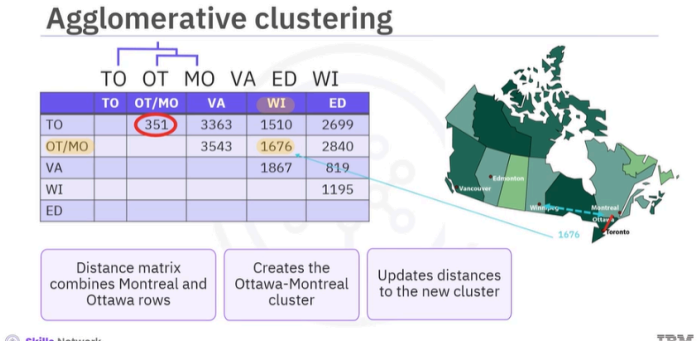


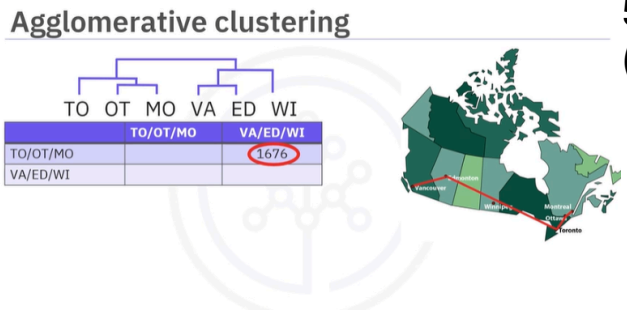
4. Agglomerative Clustering — Steps (Bottom-Up)

1. Each data point = its own cluster
2. Compute distance matrix (distances between all points)
3. Merge **closest two clusters**
4. Update distance matrix
5. Repeat until desired number of clusters or only one cluster remains
6. Visualize using **dendrogram**









5. Divisive Clustering — Steps (Top-Down)

1. Begin with **one big cluster**
2. Split into **two child clusters**
3. Recursively split each cluster
4. Stop when minimum size/criterion is met

---

6. Example (Cities in Canada)

- Use **distance matrix** (e.g., flight distances)
- Agglomerative algorithm:
  - Merge the **closest** cities first (e.g., Montreal–Ottawa)
  - Continue merging the nearest clusters
  - Draw a **dendrogram** to show hierarchy

⭐ Key Takeaways

- **Clustering = group unlabeled data**
- **K-means** = partition-based, fast, but works on simple shapes
- **DBSCAN** = good for irregular shapes and noise
- **Hierarchical clustering** = builds tree, useful for understanding relationships
- **Agglomerative**: merge upward
- **Divisive**: split downward

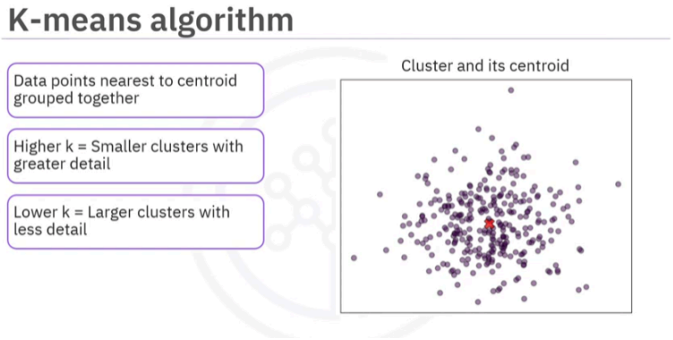
K-Means Clustering

1. What is K-Means?

- **Unsupervised**, **centroid-based**, **iterative** clustering algorithm.
- Divides data into **k non-overlapping clusters**.
- Each cluster has a **centroid** (mean of all points in that cluster).
- Goal:

  Minimize within-cluster variance (points near centroid)

  Maximize between-cluster distance (centroids far apart)



---

2. How K-Means Works (Algorithm Steps)

Step 1 — Initialization

- Choose **k**, the number of clusters.
- Randomly select **k initial centroids** (can be random points or actual data points).

Step 2 — Assignment Step

- Compute distance from each point → each centroid.
- Assign each point to the **nearest centroid**.

Step 3 — Update Step

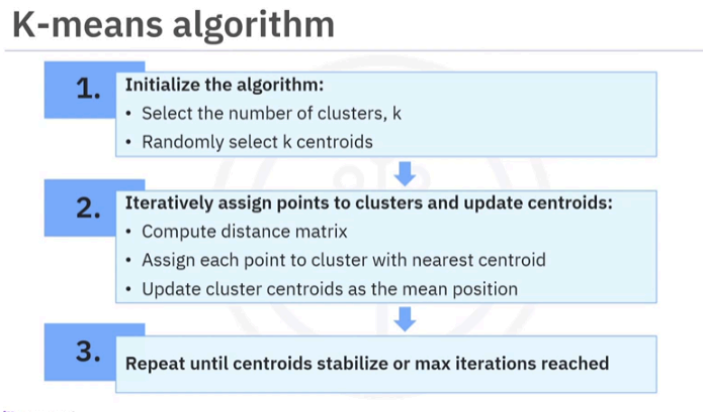
- Recompute centroid of each cluster = **mean of all points** in that cluster.

Step 4 — Repeat

- Keep repeating assignment + update
- Stop when:
  - Centroids stop changing (converge), OR
  - Max iterations reached.

Convergence

- Happens when centroid movement becomes 0 or extremely small.
- Converges quickly.



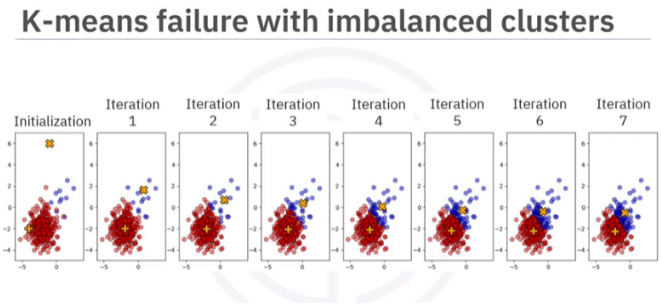
3. Behavior & Observations (Experiments)

A. Balanced clusters (same size)

- K-Means separates them well.

B. Imbalanced clusters (one large, one tiny)

- Small cluster centroid **gets pulled** toward larger cluster.
- K-Means performs poorly.
- Reason: K-Means assumes **clusters are about equal size**.

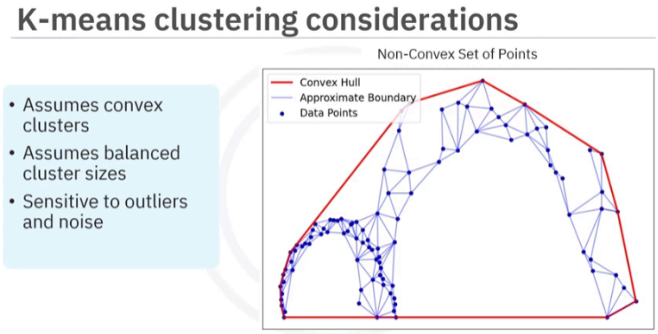


C. Non-convex clusters (e.g., moon shape)

- K-Means fails because it assumes **convex** (spherical) clusters.

D. Noise & Outliers

- Outliers distort centroid → bad performance (mean is sensitive to outliers).



K-means optimization

Goal: Minimize within-cluster sum of squares:

$$\sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

- K = Number of clusters
- $C_i = i^{\text{th}}$ cluster
- x = Data point
- μ_i = Centroid of cluster C_i
- $\|x - \mu_i\|^2$ = Squared distance between x and its cluster centroid

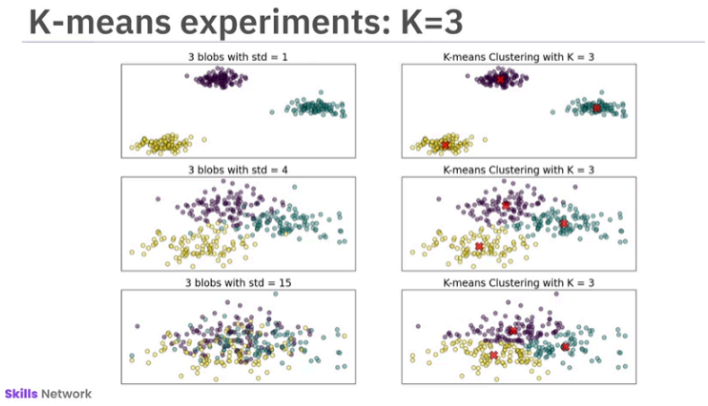
4. Standard Deviation Experiment

Dataset: 3 blobs, different spreads (std = 1, 4, 15)

- **Std = 1:** Clear clusters → K-Means performs very well.
- **Std = 4:** Some overlap → some misclassification.
- **Std = 15:** Blobs overlap completely → no clustering algorithm can separate properly.

Observation:

- As standard deviation increases → **centroids move closer** → clusters merge.

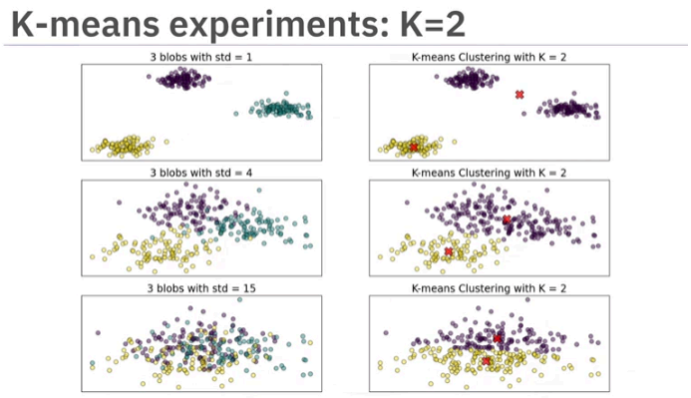


5. Case: Wrong k Value

If k is LOWER than actual classes

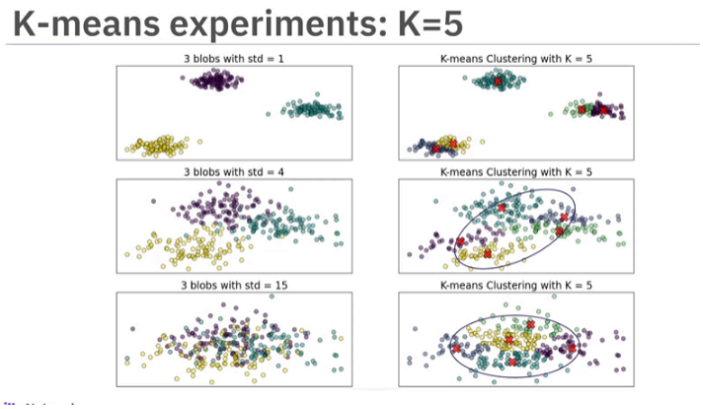
Example: 3 true classes but k = 2

- K-Means merges two blobs into one cluster.
- Centroid placed in between the merged groups.



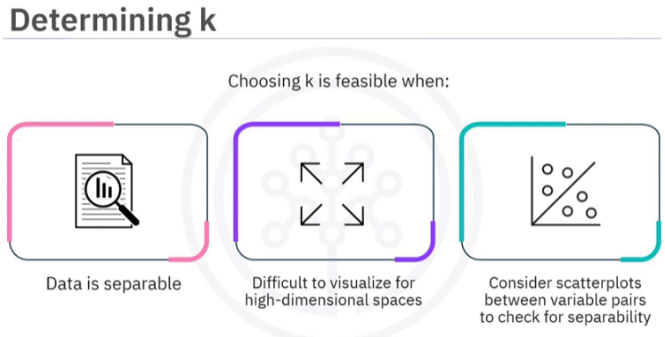
If k is HIGHER than actual classes

- K-Means artificially splits a true cluster into multiple smaller clusters.
- Results become meaningless.



7. How to Choose k (Number of Clusters)

Choosing k is **difficult** especially for high-dimensional data (cannot visualize).



Heuristic methods:

1. Elbow Method

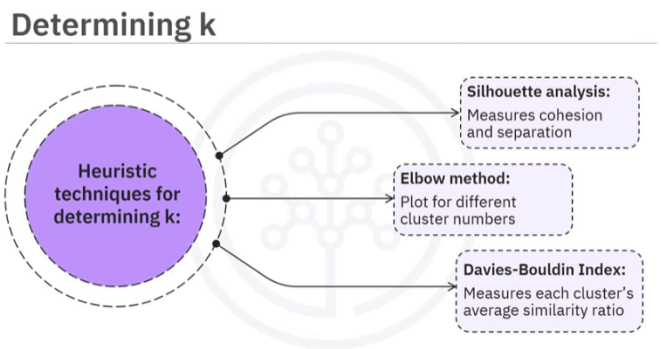
- Plot objective function vs k
- Choose the k at the “elbow” (point after which improvement slows).

2. Silhouette Score

- Measures:
  - **Cohesion**: how close a point is to its own cluster
  - **Separation**: how far it is from other clusters
- Score ranges from –1 to 1 (higher = better clusters)

3. Davies–Bouldin Index

- Measures cluster similarity
- Lower score = better clustering



Key Takeaways

- K-Means is **fast**, **scalable**, and widely used.
- Assumes:

  ✓ spherical/convex clusters

  ✓ clusters of similar size

  ✘ sensitive to outliers

  ✘ fails on irregular shapes

✔️ works well when clusters are well-separated

- Must carefully choose **k** using heuristic methods.

DBSCAN and HDBSCAN Clustering

1. What is DBSCAN?

DBSCAN = Density-Based Spatial Clustering of Applications with Noise

It is a **density-based** clustering algorithm that:

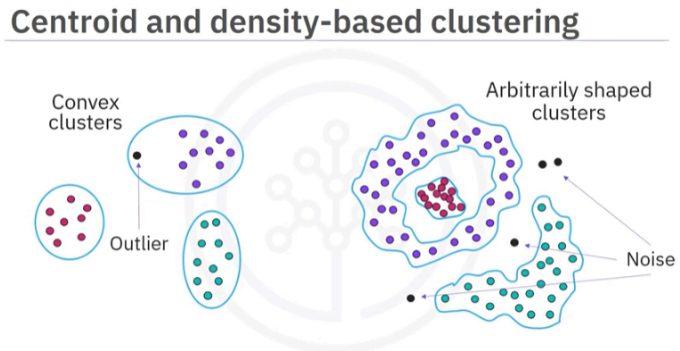
- Forms clusters based on areas of **high point density**.
- Identifies **arbitrary shapes** (not restricted to spheres like K-Means).
- Automatically detects **noise** (outliers).
- Does **not** require knowing the number of clusters beforehand.
- User provides density value

---

2. Why Density-Based Clustering?

Centroid-based algorithms (like K-Means):

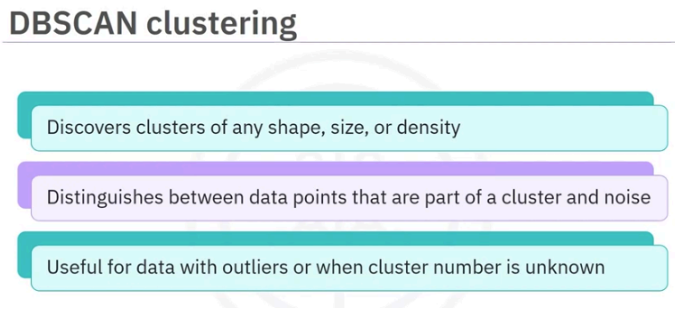
- Create **spherical/convex** clusters only.
- Force every point into a cluster → even outliers get added incorrectly.
- Fail on irregular shapes (moons, spirals).
- Fail when cluster densities differ.



Real-world data often contains:

- Noise
- Irregular shapes
- Unequal densities

DBSCAN handles these better.



---

3. DBSCAN Parameters

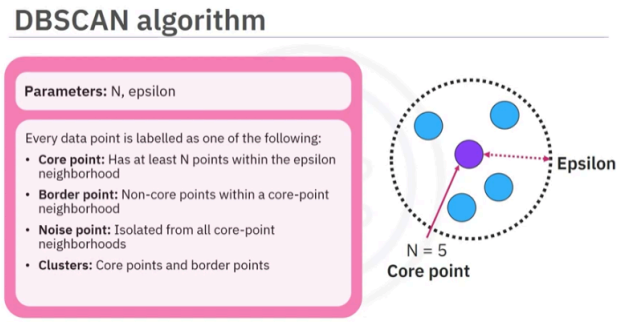
Two user-defined parameters:

1. **ε (epsilon):**

   Radius of a neighborhood around a point.

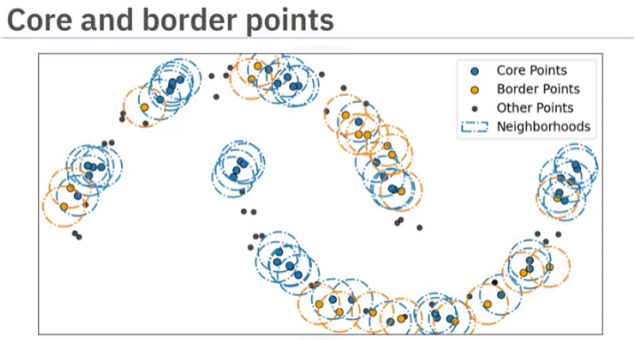
2. **minPts (n):**

   Minimum number of points required inside epsilon to form a dense region.



---

4. Types of Points in DBSCAN



Each data point becomes one of three:

(a) Core Point

- Has **≥ minPts points** in its ε-neighborhood.
- Forms the "heart" of a dense cluster.

(b) Border Point

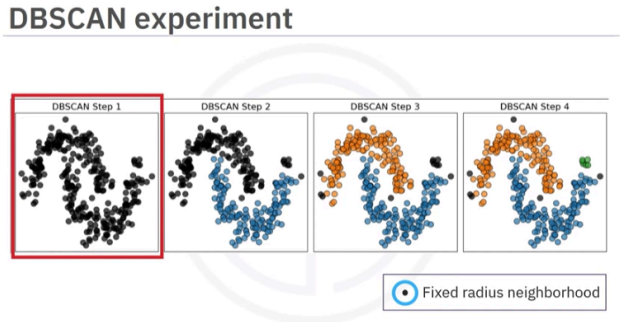
- Lies **inside** a core point's ε-neighborhood.
- But **itself does not satisfy** minPts.

(c) Noise Point

- Not a core point.
- Not in any core point's neighborhood.
- Considered **outlier**.

---

5. How DBSCAN Works (Algorithm Steps)



1. Choose ε and minPts.

2. For each point:
    - Check how many points fall within ε.
    - Classify point as **core**, **border**, or **noise**.
3. **Grow clusters from core points**:
    - Add all points in its ε-neighborhood.
4. Border points join the cluster but do **not** expand it.
5. Unassigned points = **noise**.

DBSCAN is not iterative → once a point is labeled, it stays that way.

---

6. Strengths of DBSCAN

- Finds **arbitrary-shaped clusters** (e.g., moons, curves).
- Detects **noise** naturally.
- Works well when number of clusters is **unknown**.
- Good for **spatial/geographical data**.

---

7. Weaknesses of DBSCAN

- Very **sensitive to ε value**.
- Struggles when:
    - clusters have **varying densities**
    - datasets are **high dimensional**

HDBSCAN

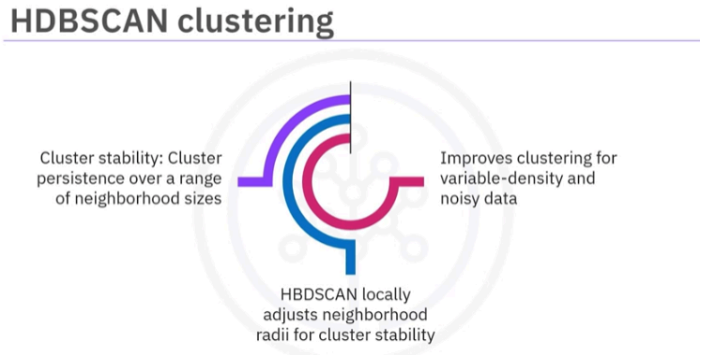
1. What is HDBSCAN?

Hierarchical Density-Based Spatial Clustering of Applications with Noise

It improves DBSCAN by:

- Removing the need to choose ε.

- Handling variable-density clusters better.
- Being less sensitive to noise.



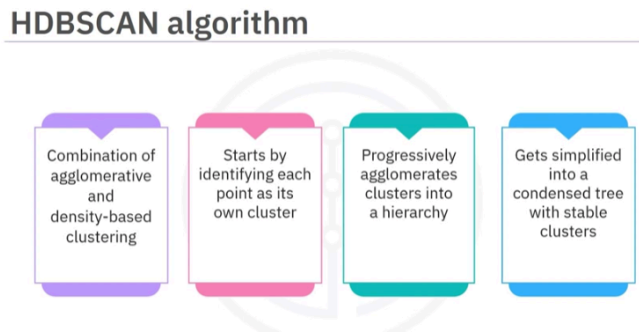
2. Key Idea: Cluster Stability

HDBSCAN checks how long a cluster remains the same while changing density thresholds.

More stable clusters = more meaningful clusters.

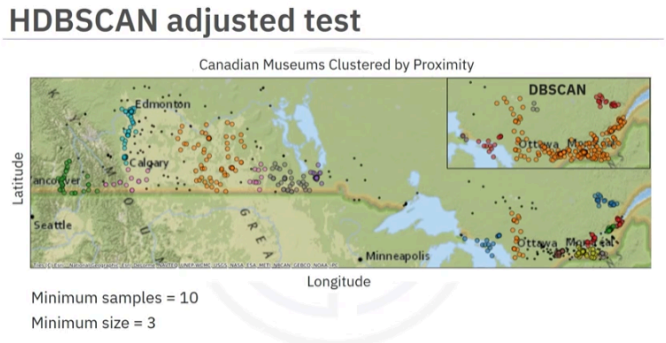
3. How HDBSCAN Works (Conceptually)

1. Start with **each point as its own cluster**.
2. Gradually **merge clusters** based on increasing neighborhood size.
3. Build a **hierarchical tree (cluster tree)**.
4. Convert it into a **condensed tree**:
    - Keep stable clusters.
    - Remove unstable/noisy ones.
5. Final clusters = the most stable branches.



4. Advantages Over DBSCAN

- No need for **ε**.
- Automatically adapts to **local densities**.
- More robust in noisy / imbalanced data.
- Finds more **fine-grained** clusters.



5. Comparison Example (Museums Data)

- DBSCAN lumped a dense city region into one giant cluster because ε couldn't adapt.
- **HDBSCAN** split that region into meaningful subclusters by adapting its radius.

6. Key Takeaways

- **DBSCAN** works on density + can find arbitrary shapes + identifies noise.

- **DBSCAN weaknesses:** fixed ε, struggles with variable densities.
- **HDBSCAN** removes need for ε, adapts to density variations, more stable.
- **Cluster stability** = how long a cluster persists across density thresholds.

Clustering, Dimension Reduction, and Feature Engineering

1. How Clustering, Dimension Reduction, and Feature Engineering Work Together

These three techniques complement each other to:

- Improve **model performance**
- Enhance **interpretability**
- Reduce **computation**
- Simplify **data structure**

---

2. Dimension Reduction (DR)

Why DR is needed

- High-dimensional data (Data with very large number of features eg Images, NLP Text, Genomics Data, Tabular data with feature expansion) makes points **sparse** and **far apart**.
- Distance-based clustering (K-Means, DBSCAN) struggles because similarity becomes unclear.
- DR reduces the number of features while keeping important information.

Common DR Methods

- **PCA (Principal Component Analysis)**
- **t-SNE**
- **UMAP**

Benefits of DR

- Makes clustering easier & more accurate.
- Makes visualization possible in **2D or 3D**.
- Reduces computation time.
- Removes noise and redundant features.

---

3. Example: Face Recognition with PCA (Eigenfaces)

Steps:

1. Take unlabeled face dataset.
2. Apply PCA → extract **top 150 eigenfaces** from 966 images.
3. These 150 eigenfaces form an **orthonormal basis**.
4. Project each face into this 150-dimensional PCA space.
5. Train an **SVM classifier** to recognize faces.

Why it works

- PCA keeps only important face patterns (edge, contours, lighting).
- Removes redundant pixel-level noise.
- Greatly reduces computational cost.

---

4. Clustering + Dimensionality Reduction

Reason

- Clustering in high-dimensions can't be visualized directly.
- DR methods (PCA, t-SNE, UMAP) project clusters into 2D/3D for:
  - Better interpretation
  - Clear visualization
  - Understanding cluster separation

---

5. Using Clustering for Feature Selection

Clustering can also be applied to **features**, not just data points.

How it helps

- Cluster similar features → identify redundant ones.
- Select **one representative feature** from each cluster.
- Reduces total features → improves model performance.

Example given

- 5 features generated with means (1, 5, 10).
- Features 1, 2, 3 have **same mean & variance** → redundant.
- K-means with **k = 3** groups features into clusters:
  - Cluster 1 = redundant features (1–3)
  - Feature 4 and feature 5 stand out separately

This is both:

- **Feature selection**
- **Dimension reduction**

---

6. Key Takeaways

Clustering

- Finds patterns and subgroups.
- Helps with feature engineering (identify subgroup-specific transformations).

Dimension Reduction

- Reduces feature count.
- Helps clustering algorithms perform better.
- Necessary for visualization.

Feature Engineering

Creating New useful features or transform existing features to help the model learn better

1. Feature Creation

- New features from existing data
  - Ratios (e.g., income/expense)
  - Feature interactions (×, +)
  - Text features (word count, TF-IDF)
  - Date/time features (day, month, hour)

---

2. Feature Transformation

- Adjust feature scale or distribution
  - Log transform (fix skew)
  - Normalization (0–1)
  - Standardization (Z-score)

---

3. Encoding Categorical Data

- Convert categories to numbers
  - One-hot encoding (nominal)
  - Label encoding (ordinal)
  - Target or frequency encoding (high-cardinality)

4. H**andling Missing Values**

- Uses clustering to detect redundant or similar features.
- Selects or transforms features to improve the model.

Dimension Reduction Algorithms

Why Dimension Reduction?

- Reduce number of features while **keeping important information**
- Helps in **visualization**, **noise reduction**, and **model performance**
- Useful for **high-dimensional** datasets (images, text, embeddings)

---

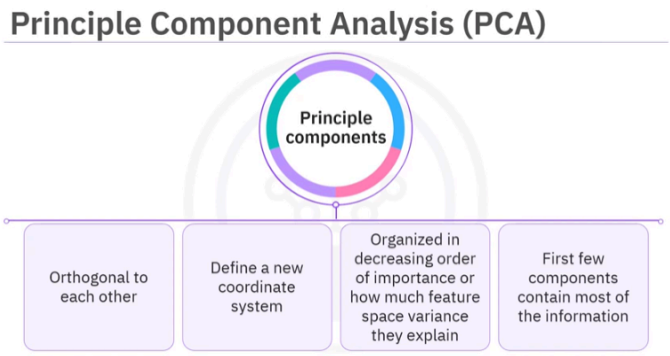
1. PCA (Principal Component Analysis)

Type: Linear

Goal: Find new axes (principal components) that capture the **maximum variance**

Key Points

- Transforms data into **uncorrelated** components
- Components are **ordered** by how much variance they explain
- Works well when data is **linearly correlated**
- Reduces noise and simplifies data



---

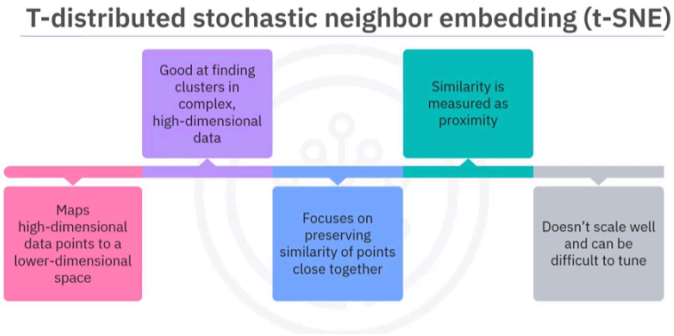
2. t-SNE (t-Distributed Stochastic Neighbor Embedding)

Type: Nonlinear

Goal: Preserve **local structure** (points that are close stay close)

Key Points

- Excellent for **visualizing clusters** in 2D/3D
- Common in image + text embeddings
- Sensitive to hyperparameters
- **Does not scale well** (slow for large datasets)
- Preserves **local similarity**, not global structure



---

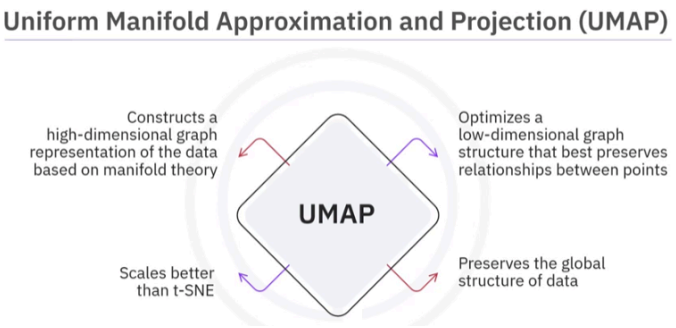
3. UMAP (Uniform Manifold Approximation and Projection)

Type: Nonlinear, manifold-based

Goal: Preserve **local + some global** structure

Key Points

- Faster and more scalable than t-SNE
- Better at maintaining overall data structure
- Often used before clustering
- Uses graph-based manifold approximation



Comparison Summary (Super Short)

Algorithm	Linear?	Preserves	Speed	Best Use
PCA	Yes	Global variance	Very fast	Noise reduction, linear data
t-SNE	No	Local structure	Slow	Cluster visualization
UMAP	No	Local + some global	Fast	Large datasets, clustering

Building Unsupervised Learning Models

Unsupervised learning models

Model Name	Brief Description	Code Syntax
UMAP	UMAP (Uniform Manifold Approximation and Projection) is used for dimensionality reduction.**Pros:** High performance, preserves global structure.**Cons:** Sensitive to parameters.**Applications:** Data visualization, feature extraction.**Key hyperparameters:** • **n_neighbors:** Controls the local neighborhood size (default = 15). • **min_dist:** Controls the minimum distance between points in the	1. from umap.umap_ import UMAP 2. umap = UMAP(n_neighbors=15, min_dist=0.1, n_components=2)

Model Name	Brief Description	Code Syntax
	embedded space (default = 0.1). • **n_components:** The dimensionality of the embedding (default = 2).	
t-SNE	t-SNE (t-Distributed Stochastic Neighbor Embedding) is a nonlinear dimensionality reduction technique.**Pros:** Good for visualizing high-dimensional data.**Cons:** Computationally expensive, prone to overfitting.**Applications:** Data visualization, anomaly detection.**Key hyperparameters:** • **n_components:** The number of dimensions for the output (default = 2). • **perplexity:** Balances attention between local and global aspects of the data (default = 30). • **learning_rate:** Controls the step size during optimization (default = 200).	1. from sklearn.manifold import TSNE 2. tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)
PCA	PCA (principal component analysis) is used for linear dimensionality reduction.**Pros:** Easy to interpret, reduces noise.**Cons:** Linear, may lose information in nonlinear data.**Applications:** Feature extraction, compression.**Key hyperparameters:** • **n_components:** Number of principal components to retain (default = 2). • **whiten:** Whether to scale the components (default = False). • **svd_solver:** The algorithm to	1. from sklearn.decomposition import PCA 2. pca = PCA(n_components=2)

Model Name	Brief Description	Code Syntax
	compute the components (default = 'auto').	
DBSCAN	DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm.**Pros:** Identifies outliers, does not require the number of clusters.**Cons:** Difficult with varying density clusters.**Applications:** Anomaly detection, spatial data clustering.**Key hyperparameters:** • **eps:** The maximum distance between two points to be considered neighbors (default = 0.5). • **min_samples:** Minimum number of samples in a neighborhood to form a cluster (default = 5).	1. from sklearn.cluster import DBSCAN 2. dbscan = DBSCAN(eps=0.5, min_samples=5)
HDBSCAN	HDBSCAN (Hierarchical DBSCAN) improves on DBSCAN by handling varying density clusters.**Pros:** Better handling of varying densities.**Cons:** Can be slower than DBSCAN.**Applications:** Large datasets, complex clustering problems.**Key hyperparameters:** • **min_cluster_size:** The minimum size of clusters (default = 5). • **min_samples:** Minimum number of samples to form a cluster (default = 10).	1. import hdbscan 2. clusterer = hdbscan.HDBSCAN(min_cluster_size=5)
K-Means clustering	K-Means is a centroid-based clustering algorithm that groups data into k clusters.**Pros:** Efficient, simple to implement.**Cons:** Sensitive to initial	1. from sklearn.cluster import KMeans 2. kmeans = KMeans(n_clusters=3)

Model Name	Brief Description	Code Syntax
	cluster centroids.**Applications:** Customer segmentation, pattern recognition.**Key hyperparameters:** • **n_clusters:** Number of clusters (default = 8). • **init:** Method for initializing the centroids ('k-means++' or 'random', default = 'k-means++'). • **n_init:** Number of times the algorithm will run with different centroid seeds (default = 10).	

Associated fuctions used

Method	Brief Description	Code Syntax
make_blobs	Generates isotropic Gaussian blobs for clustering.	1. from sklearn.datasets import make_blobs 2. X, y = make_blobs(n_samples=100, centers=2, random_state=42)
multivariate_normal	Generates samples from a multivariate normal distribution.	1. from numpy.random import multivariate_normal 2. samples = multivariate_normal(mean=[0, 0], cov=[[1, 0], [0, 1]], size=100)
plotly.express.scatter_3d	Creates a 3D scatter plot using Plotly Express.	1. import plotly.express as px 2. fig = px.scatter_3d(df, x='x', y='y', z='z') 3. fig.show()Copied!Wrap Toggled!
geopandas.GeoDataFrame	Creates a GeoDataFrame from a Pandas DataFrame.	1. import geopandas as gpd 2. gdf = gpd.GeoDataFrame(df, geometry='geometry')Copied!Wrap Toggled!
geopandas.to_crs	Transforms the coordinate reference	

Method	Brief Description	Code Syntax
	system of a GeoDataFrame.	1. gdf = gdf.to_crs(epsg=3857)Copied!Wrap Toggled!
contextily.add_basemap	Adds a basemap to a GeoDataFrame plot for context.	1. import contextily as ctx 2. ax = gdf.plot(figsize=(10, 10)) 3. ctx.add_basemap(ax)Copied!Wrap Toggled!
pca.explained_variance_ratio_	Returns the proportion of variance explained by each principal component.	1. from sklearn.decomposition import PCA 2. pca = PCA(n_components=2) 3. pca.fit(X) 4. variance_ratio = pca.explaine