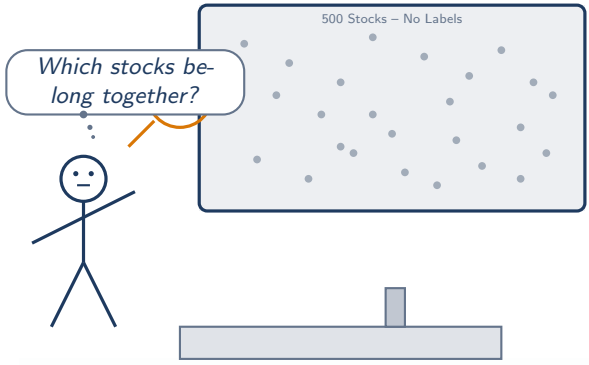
Unsupervised Learning: From K-Means to Gaussian Mixtures

Data Science with Python – BSc Course (Supplementary Lecture)

Data Science Program

BSc Course

90 Minutes



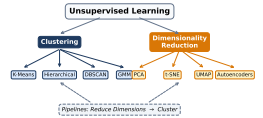
Unsupervised learning finds the groups you didn't know were there.

What Methods Make Up Unsupervised Learning?

Unsupervised learning (discovering structure without labels) splits into two branches:

- **Clustering** — group similar observations: K-Means, Hierarchical, DBSCAN, HDBSCAN, GMM
- **Dim. Reduction** — compress features: PCA, t-SNE, UMAP

GMM spans both: clustering *and* density estimation.
Every method answers one question: *How do you find patterns without labels?*



Read the chart:

- The tree splits UL into clustering (navy) and reduction (amber)
- Dashed borders mark methods spanning multiple branches
- Complexity increases left to right: K-Means is simplest

Every method on this map gets at least one slide. We start simple and add complexity.

Why Should You Learn Unsupervised Learning?

Industry scale:

- UL market: **USD 14.35 billion** (2024), projected USD 38.72B by 2032 (CAGR 13.45%)
- Banking, Financial Services & Insurance (BFSI) leads with **33.6%** market share
- NLP (Natural Language Processing) captures 40% of tech share — topic modeling, document clustering, embedding search are all unsupervised

Top use cases:

1. Customer segmentation (K-Means, GMM)
2. Anomaly and fraud detection (DBSCAN, isolation forests)
3. Feature extraction and preprocessing (PCA, autoencoders)
4. Data visualization (t-SNE, UMAP)

UL is not an alternative to supervised learning — it is a **complement**. Most production pipelines include an unsupervised step.

Source: market.us (2025). BFSI = Banking, Financial Services & Insurance.

What Will You Be Able to Do After This Lecture?

By the end of this 90-minute session you will be able to:

1. **Cluster** data using K-Means, hierarchical, DBSCAN, and GMM — and choose the right method for each situation.
2. **Reduce** dimensionality with PCA and visualize high-dimensional data using t-SNE and UMAP.
3. **Validate** clustering results using the elbow method, silhouette scores, and BIC (Bayesian Information Criterion).
4. **Build** a leakage-free sklearn Pipeline for unsupervised workflows.
5. **Avoid** common pitfalls: wrong distance metric, ignoring feature scales, and misinterpreting t-SNE cluster distances.

These five verbs map to Bloom's levels: apply, apply, evaluate, create, analyze.

How Do You Measure Similarity Between Two Data Points?

Clustering requires a **distance metric** (a rule for measuring how close two points are).

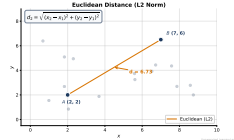
Euclidean distance (straight-line distance):

$$d(x, y) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

Example: Stock A = [0.05, 0.12], Stock B = [0.08, 0.15].

$$d = \sqrt{(0.05-0.08)^2 + (0.12-0.15)^2} = \sqrt{0.0018} = 0.042$$

Euclidean assumes all features are on the same scale — **always standardize first**.



Read the chart:

- Two points connected by a dashed line
- Length = Euclidean distance ("as the crow flies")
- Unscaled axes would distort the measurement

Euclidean distance is sklearn's KMeans default. It works well when features are continuous and on similar scales.

When Should You Use a Different Distance Metric?

Manhattan distance ("taxi-cab" distance):

$$d(x, y) = \sum_{i=1}^{p} |x_i - y_i|$$

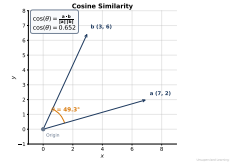
Robust to outliers (no squaring). Use for sparse data.

Cosine similarity (angle, not magnitude):

$$\cos(\theta) = \frac{x \cdot y}{\|x\| \, \|y\|}$$

Ignores portfolio size, focuses on weight proportions.

Finance example: Two portfolios with identical stock *weights* but different total capital: cosine similarity = 1, Euclidean distance is large.



Read the chart:

- The arc shows the cosine angle between two vectors
- Cosine measures direction, not distance
- Two parallel vectors score 1 regardless of length

K-Means uses Euclidean by default. Switch to cosine for text data (TF-IDF vectors). Switch to Manhattan for data with outliers.

How Does K-Means Find Clusters Without Labels?

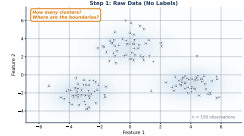
Input: data X (standardized), number of clusters K.

1. Pick K random initial centroids (cluster centers) $\mu_1, \ldots, \mu_K$
2. **Assign** each point to its nearest centroid
3. **Update** each centroid as the mean of its points
4. Repeat steps 2–3 until assignments stop changing

Objective — minimize inertia (within-cluster sum of squares):

$$J = \sum_{k=1}^{K} \sum_{x \in C_k} \|x - \mu_k\|^2$$

```
KMeans(n_clusters=K, n_init=10,
random_state=42)
```



Read the chart:

- Raw scatter — no colors yet
- K-Means will assign colors by iterating assign-update
- Natural groupings suggest K=3 might work

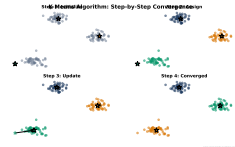
K-Means is 67 years old (Lloyd, 1957) and still the most-used clustering algorithm. As you learned in L29, it converges in 5–20 iterations.

How Does K-Means Converge Step by Step?

Iteration 1: random centroids, messy assignments.
Iteration 4: centroids migrate toward natural cluster centers.
Iteration 7: stable — centroids barely move, assignments fixed.

Convergence guarantee: inertia J decreases (or stays the same) at every iteration, so the algorithm always stops.

Warning: K-Means finds a **local minimum**, not the global one. That is why sklearn runs it `n_init=10` times with different random starts and keeps the best result.



Read the chart:

- Three snapshots: early (messy), middle, late (clean)
- Arrows show centroids moving to final positions
- By iteration 7 the cluster boundaries are clear

K-Means always converges, but may converge to a bad solution. Running with multiple random starts guards against this.

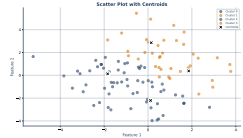
Can K-Means Group 200 Stocks Into Risk Buckets?

Problem: Group 200 stocks by risk-return characteristics for portfolio construction.
Features (all standardized): annualized return, volatility, Sharpe ratio (return per unit risk), beta (sensitivity to the market).

Result (K=4):

1. Defensive low-volatility
2. Growth high-return
3. Speculative high-volatility
4. Market-neutral

Pick one stock from each cluster for instant diversification (spreading risk across different types of assets).



Read the chart:

- Each dot = one stock; colors = cluster labels
- Axes show return vs. volatility
- Clusters map to risk-return quadrants

In practice, always standardize features before clustering. Raw features on different scales produce meaningless Euclidean distances.

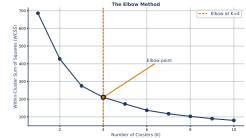
How Do You Choose the Right Number of Clusters?

K-Means requires K as input, but the "right" K is unknown.

Elbow method: plot inertia J vs. K (from 1 to 10). Look for the **elbow** — the point where adding more clusters gives diminishing returns.

Example: J drops sharply from $K{=}1$ to $K{=}3$, then flattens. The elbow is at $K{=}3$.

Warning: the elbow is often ambiguous. There may be no clear kink. Use the elbow as a starting point, then validate with the silhouette score (next slide).



Read the chart:

- x-axis = K, y-axis = inertia
- Inertia always decreases as K grows
- The "elbow" marks where marginal gain slows

If the elbow is ambiguous, try the silhouette score. No single metric gives the "true" K — domain knowledge matters.

How Good Are Your Clusters, Really?

For each point i, compute:

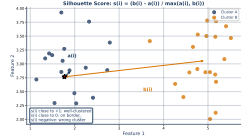
- $a(i)$ = average distance to points in the **same** cluster
- $b(i)$ = minimum average distance to points in any **other** cluster

Silhouette score:

$$s(i) = \frac{b(i) - a(i)}{\max\big(a(i),\ b(i)\big)} \in [-1,\ +1]$$

Example: $a(i)$=1.2, $b(i)$=3.8.
$s(i) = (3.8-1.2)/3.8 = 0.68$ — well-clustered.

Rules of thumb: > 0.5 = strong, 0.25–0.5 = acceptable, < 0.25 = weak.



Read the chart:

- Each bar = one point; width = silhouette score
- Colors group points by cluster
- The dashed line = average score; higher is better

Use elbow AND silhouette together. A KM-DBSCAN hybrid improved accuracy by 15% over plain K-Means on banking data (Journal La Multiapp, 2025).

What If You Don't Know How Many Groups Exist?

Situation: You have 50 assets and want to find groups for portfolio construction.

Complication: You don't know whether the "right" grouping has 3 sectors, 5 risk buckets, or 10 micro-clusters. K-Means *forces* you to choose K before seeing the data.

Question: Can you build all possible groupings at once and choose later?

Agglomerative clustering (bottom-up hierarchical clustering):

1. Start with each point as its own cluster
2. Merge the two closest clusters
3. Repeat until only one cluster remains

This produces a **dendrogram** (tree diagram) that encodes every possible grouping. Cut the tree at any height to get K clusters for any K.

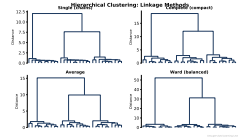
Hierarchical clustering trades speed for flexibility: $O(n^2)$ memory, $O(n^3)$ time. Use K-Means for $>10\,000$ points.

How Do You Measure the Distance Between Two Clusters?

When merging clusters, the **linkage method** (rule for measuring inter-cluster distance) determines the shape of the result:

- **Single:** minimum distance between any two points across clusters. Finds elongated chains.
- **Complete:** maximum distance. Compact, spherical clusters.
- **Average:** mean of all pairwise distances. A compromise.
- **Ward:** minimizes increase in total within-cluster variance. Most similar to K-Means.

Default recommendation: Ward for general use. Single linkage for detecting outliers or elongated shapes.



Read the chart:

- Same data, four linkage methods
- Single produces "chaining"; Ward produces spheres
- Linkage choice matters as much as K

sklearn default is Ward. Use `AgglomerativeClustering(n_clusters=K, linkage='ward')`**. As you learned in L30, Ward is most similar to K-Means.**

How Do You Read a Clustering Family Tree?

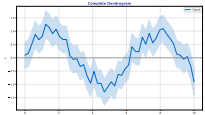
A **dendrogram** (tree diagram) shows the merge history.

- **Horizontal axis:** data points (leaves)
- **Vertical axis:** merge distance (height)

Points merged at low height are very similar; points merged at high height are dissimilar.

How to choose K: draw a horizontal line across the dendrogram. The number of vertical branches it crosses $= K$.

Rule of thumb: cut where there is a large **gap** between merge heights (analogous to the elbow method).



Read the chart:

- Each leaf = one data point; bars = merges
- Bar height = distance between merged clusters
- A horizontal cut at height 5 gives 3 clusters

A dendrogram is to hierarchical clustering what a scatterplot is to regression — the essential diagnostic tool.

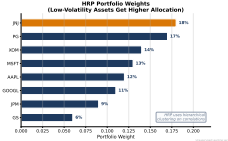
Can a Dendrogram Build a Better Portfolio Than Markowitz?

Problem: Markowitz mean-variance optimization is unstable — small estimation errors in the covariance matrix produce wildly different portfolios.

HRP (Hierarchical Risk Parity, Lopez de Prado 2016):

1. Cluster assets by correlation using a dendrogram
2. Quasi-diagonalize the covariance matrix
3. Allocate weights by recursive bisection along the tree

Backtest (2005–2025): Sharpe ratio **0.4711**, max drawdown (largest peak-to-trough loss) −60%, cumulative return 1,299.67%.



Read the chart:

- HRP distributes weights more evenly across asset clusters
- No single asset dominates the portfolio
- Diversification is encoded in the tree structure

Source: Lopez de Prado (2016), SSRN 2708678. Named "Quant of the Year." HRP is in riskfolio-lib and PyPortfolioOpt.

When Should You Use K-Means vs. Hierarchical?

Criterion	K-Means	Hierarchical
Requires K in advance?	Yes	No (cut later)
Cluster shape	Spherical only	Linkage-dependent
Scalability	$O(nKt)$, millions OK	$O(n^2)$ mem, $O(n^3)$ time
Reproducibility	Random init	Deterministic
Output	Flat partition	Full tree (dendrogram)
Best for	Large data, known K	Small data, HRP

In practice:

- Use K-Means when you *know* K and have $>10\,000$ points
- Use hierarchical when K is unknown, $n < 5{,}000$, or you need the tree structure (e.g., HRP portfolio construction)

K-Means is fast and simple; hierarchical is flexible and informative. Choose based on data size and whether you know K.

Why Do 200 Features Break Your Clustering?

Situation: You have daily returns for 200 stocks and want to understand their co-movement structure.

Complication: 200 features means 200 axes. You cannot visualize it. Pairwise correlations give a 200×200 matrix — 19,900 numbers. K-Means in 200-dimensional space is unreliable because all pairwise distances converge to roughly the same value.

Question: Can you reduce 200 features to 5–10 "super-features" that capture most of the information?

That is **dimensionality reduction**. PCA (Principal Component Analysis — a method that finds the directions of maximum variance) is the standard starting point.

The curse of dimensionality: in 200 dimensions, the nearest and farthest neighbors have nearly the same distance. Clustering fails.

Why Do Distances Lose Meaning in High Dimensions?

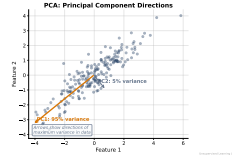
In 2D: data fills an area. **In 100D:** data concentrates near the surface of a hypersphere.

Distances converge: the ratio of maximum to minimum distance approaches 1 as the number of features p grows.

K-Means uses Euclidean distance. When all distances are similar, K-Means *cannot* distinguish clusters.

Rule of thumb: you need $n \gg 2^p$ data points for reliable distance-based methods. With p=200, that is impossible.

Solution: reduce p to 5–10 principal components that capture 80–95% of variance.



Read the chart:

- High-dimensional data projected to 2D via PCA
- Dominant structure (clusters, trends) survives
- Some overlap = information loss, but main patterns remain

The curse of dimensionality is not theoretical — it causes real failures in clustering and classification. PCA is the standard remedy.

How Does PCA Find the Directions of Maximum Variance?

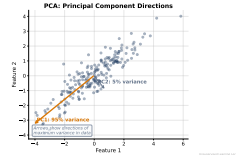
PCA finds new axes (**principal components**) that are:

1. Orthogonal (uncorrelated with each other)
2. Ordered by variance explained (PC1 captures the most)

Mathematically, PC directions are **eigenvectors** v_i (directions of maximum spread) of the covariance matrix; the **eigenvalues** λ_i are the variance along each direction.

Finance interpretation:
PC1 $\approx$ market factor (all stocks move together)
PC2 $\approx$ sector rotation
PC3 $\approx$ value vs. growth

First 3 PCs explain $\sim$**75.2%** of equity return variance.



Read the chart:

- A cloud of points with PC1 and PC2 as axes
- PC1 (horizontal) captures the greatest spread
- PC2 captures the remaining spread, perpendicular to PC1

PCA is an eigenvalue decomposition of the covariance matrix. As you learned in L31, it rotates axes to maximize variance.

When Should You Stop Adding Principal Components?

Scree plot: eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots$ plotted in order. Look for an "elbow."

Cumulative variance:

$$\text{CVR}(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$$

Keep enough components to explain 80–95% of total variance.

Kaiser criterion: keep components with $\lambda_i > 1$ (eigenvalue greater than average variance per variable in standardized data).

Example: 200 stock features. First 5 PCs explain 78%, first 10 explain 92%. Keep 10.

Read the chart:

- x-axis = component number, y-axis = eigenvalue
- First few eigenvalues dominate; the rest decay
- The "elbow" around 5–10 suggests where to cut

The scree plot for financial returns typically shows 3–5 dominant factors. Everything beyond is noise or idiosyncratic risk.

What Do Your Principal Components Actually Mean?

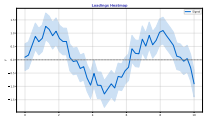
Loadings (weights of original features in each PC): $v_1 = [0.12,\ 0.08,\ -0.05,\ \ldots]$

A **loading heatmap** shows which features drive each component.

Biplot (bivariate plot): overlays data points (scores) and feature arrows (loadings) in the same PC1–PC2 space.

Finance interpretation:

- PC1 loads equally on all stocks $\Rightarrow$ "market factor"
- PC2 loads positively on tech, negatively on utilities $\Rightarrow$ "sector rotation factor"



Read the chart:

- Rows = features; columns = PCs
- Dark cells = high loadings (positive or negative)
- PC1 loads on everything (market); PC2 splits groups

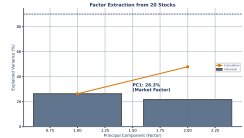
Loadings turn abstract components into interpretable factors. In finance, PC1 is almost always "the market."

How Does PCA Extract Risk Factors From 200 Stocks?

PCA for factor extraction: extract 3–5 statistical factors from 200 stock returns.
These factors are **data-driven** (unlike Fama-French, which are theory-driven).

Applications:

1. **Risk decomposition.** How much portfolio risk comes from the market factor vs. sector rotation?
2. **Anomaly detection.** A stock far from its expected PC-space position may be mispriced.
3. **Yield curve modeling.** First 3 PCs of bond yields = level, slope, curvature.



Read the chart:

- Stocks projected onto PC1 and PC2
- Same-sector stocks cluster together
- Outliers in PC space warrant investigation

PCA is the unsupervised counterpart to regression-based factor models. Both extract factors, but PCA needs no predefined definitions.

Is PCA Always Enough, or Do You Need a Neural Network?

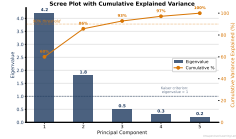
PCA is linear: it captures only linear relationships.

Autoencoders (neural networks): learn nonlinear compression via a low-dimensional "bottleneck." They capture curved manifold structures that PCA misses.

When is PCA sufficient?

- Financial returns: relationships are approximately linear
- PCA is $\sim$100$\times$ faster than autoencoders
- PCA axes are interpretable; autoencoder axes are not

When to upgrade: images, text embeddings, complex nonlinear data. Covered in L33–L36 (Deep Learning).



Read the chart:

- Cumulative variance explained vs. components kept
- Curve rises steeply, then flattens
- Dashed line at 90%: typically 5–10 components suffice

PCA is the default for dimensionality reduction in finance. Use autoencoders only when you have evidence that relationships are nonlinear.

What If Your Clusters Are Not Spherical?

DBSCAN (Density-Based Spatial Clustering of Applications with Noise): clusters are dense regions separated by sparse regions.

Two parameters:

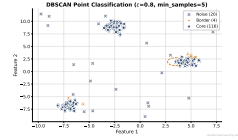
- ϵ = neighborhood radius
- MinPts = minimum points in neighborhood

Three point types:

- **Core:** $\geq$ MinPts neighbors within ϵ
- **Border:** within ϵ of a core point
- **Noise:** neither — labeled as -1

No K needed. The number of clusters is discovered automatically.

```
DBSCAN(eps=0.5, min_samples=5).fit(X)
```



Read the chart:

- Large dots = core; small dots = border
- Red X marks = noise (no cluster)
- Shaded circles show the ϵ-radius neighborhoods

DBSCAN was introduced by Ester et al. (1996) and remains the most popular density-based clustering algorithm.

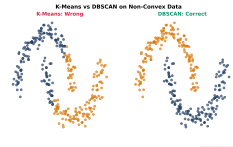
Why Does K-Means Fail on Crescent-Shaped Data?

K-Means assumes **spherical** clusters because it uses Euclidean distance to centroids.
On crescent-shaped ("two moons") or ring-shaped data, K-Means splits clusters incorrectly.

DBSCAN follows density contours regardless of shape.

Trade-off: DBSCAN requires tuning ϵ and MinPts. Poor choices produce bad results.

Practical tip: use a k-distance plot (sort distances to the k-th nearest neighbor) to estimate ϵ. The "knee" in the plot suggests a good value.
MinPts $\geq 2 \times p$ (twice the number of dimensions) is a common starting point.



Read the chart:

- Left: K-Means splits moons incorrectly
- Right: DBSCAN follows each crescent's density
- K-Means is not wrong — it solves a different problem

K-Means cuts spheres; DBSCAN follows density. Match the algorithm to the cluster shape in your data.

Can You Run DBSCAN Without Tuning Epsilon?

DBSCAN's main weakness: results are very sensitive to ϵ.

HDBSCAN (Hierarchical DBSCAN): runs DBSCAN over a *range* of ϵ values automatically. It builds a hierarchy of density levels and selects the most **stable** clusters.

Key advantages:

- Only one parameter: `min_cluster_size` (minimum cluster membership)
- Handles clusters of varying density (single ϵ cannot do this)
- Available in sklearn since v1.3 (2023)
- Optional **soft clustering** — probability-based assignments

```
HDBSCAN(min_cluster_size=15).fit(X)
```

When to use HDBSCAN over DBSCAN: clusters of varying density, or you do not want to tune ϵ.

HDBSCAN is the modern default for density-based clustering. Use DBSCAN only when you need explicit control over epsilon.

How Do You Visualize 50 Dimensions in 2D?

t-SNE (t-Distributed Stochastic Neighbor Embedding): preserves local neighborhoods when projecting to 2D.

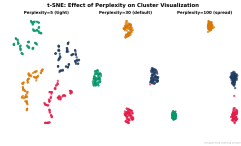
Perplexity parameter (σ): controls the effective number of neighbors considered. Typical range: 5–50.

Critical sensitivity: same data at perplexity=10 shows **15 clusters**; at perplexity=50 shows **3 clusters**.

Three warnings:

1. Distances *between* clusters are meaningless
2. Cluster sizes are meaningless
3. Different runs may look different (stochastic)

Use t-SNE for **visualization only** — never for downstream ML.



Read the chart:

- Three projections with different perplexity values
- Low perplexity: tight, many clusters
- High perplexity: fewer, merged clusters

---

t-SNE is for visualization, not analysis. Never cluster the t-SNE output. Never interpret inter-cluster distances.

Is There a Faster Alternative to t-SNE?

UMAP (Uniform Manifold Approximation and Projection, McInnes et al. 2018):

- **Faster:** $O(n \log n)$ vs. t-SNE's $O(n^2)$. Handles 5 million points in 47 minutes.
- **Better global structure:** 15–25% higher trustworthiness (a metric that measures how well neighborhoods are preserved).
- **More reproducible:** less sensitive to random initialization.
- **Usable for reduction:** unlike t-SNE, UMAP output can feed into downstream models.

Key parameter: `n_neighbors` (analogous to perplexity). Higher = more global context.

```
import umap; reducer = umap.UMAP(n_neighbors=15)
embedding = reducer.fit_transform(X)
```

Practical tip: run PCA to 30–50 dimensions first, then UMAP. This reduces runtime by ∼60%.

UMAP has largely replaced t-SNE as the default visualization method. Use t-SNE only when you specifically want the "exploded" local view.

Which Dimensionality Reduction Method Should You Start With?

	PCA	t-SNE	UMAP
Type	Linear	Nonlinear	Nonlinear
Preserves	Global variance	Local neighborhoods	Local + some global
Speed	Very fast	Slow $O(n^2)$	Fast $O(n \log n)$
Axes interpretable?	Yes	No	No
Downstream ML?	Yes	No	Yes
Best for	Baseline, interpretable	Exploring clusters	General visualization

Recommended workflow:

1. **Start with PCA** — interpretable axes, fast, identifies dominant factors
2. **Switch to UMAP** if PCA does not separate clusters visually
3. **Use t-SNE** for publication-quality cluster visualizations (fine-grained local detail)

Start with PCA for interpretability. Switch to UMAP or t-SNE when PCA does not separate clusters visually.

What If a Data Point Belongs to Multiple Clusters at Once?

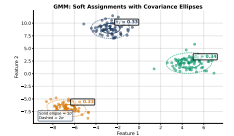
K-Means assigns each point to *exactly* one cluster. But what if a point sits between two clusters?

GMM (Gaussian Mixture Model): models data as a mixture of K Gaussian distributions, each with mean μ_k, covariance Σ_k, and weight π_k (mixing proportion).

Soft assignment: each point gets a *probability* of belonging to each cluster.

Example: A stock with 70% probability "growth" and 30% "value" — captures the ambiguity K-Means ignores.

```
GaussianMixture(n_components=K).fit(X)
```



Read the chart:

- Color gradient = cluster probability (not hard label)
- Ellipses show Gaussian components (mean + covariance)
- Boundary points have ∼50/50 probability

GMM is K-Means with uncertainty. If you need probabilities (risk assessment, portfolio allocation), use GMM. Complexity: O($n \cdot k \cdot d^2$).

How Does a Gaussian Mixture Model Learn Its Parameters?

EM (Expectation-Maximization) is a two-step iterative algorithm:

1. **E-step (Expectation):** Compute the probability that each point belongs to each Gaussian component (soft assignments). Similar to K-Means "assign."
2. **M-step (Maximization):** Update each component's μ_k, Σ_k, π_k to maximize the likelihood of the data. Similar to K-Means "update centroids."

Iterate E–M until convergence (log-likelihood stabilizes).

Key insight: K-Means is a **special case** of EM where covariances are identity matrices and assignments are hard (0 or 1).

Convergence: EM always increases log-likelihood, so it always converges (to a local maximum, same caveat as K-Means).

EM is to GMM what the assign-update loop is to K-Means. If you understand K-Means, you understand 80% of EM.

How Do You Choose the Number of Gaussian Components?

The elbow method works for K-Means, but GMM needs a **likelihood-based criterion**.

BIC (Bayesian Information Criterion):

$$\text{BIC} = -2 \ln L + k \ln n$$

where L = maximized likelihood, k = number of parameters, n = number of data points. **Penalizes complexity.**

AIC (Akaike Information Criterion):

$$\text{AIC} = -2 \ln L + 2k$$

Less penalty for complexity than BIC.

Lower BIC/AIC = better model. Plot BIC vs. number of components; choose the minimum.

Example: BIC minimized at K=3. Adding a 4th component increases BIC (overfitting penalized).

BIC and AIC balance fit vs. complexity. BIC is the safer choice for clustering — it penalizes overfitting more.

Why Does Your Notebook Model Fail in Production?

Situation: You scale your features (StandardScaler), run K-Means, and evaluate with the silhouette score.

Complication: If you scale on the *entire* dataset — including the evaluation data — you have **data leakage**. The scaler "sees" information from data it should not have access to. The silhouette score is optimistically biased.

Question: How do you ensure that each step only uses information it is allowed to?

Answer: Use an sklearn **Pipeline** (a container that chains preprocessing and modeling steps). The pipeline ensures that cross-validation, train/test splits, and evaluation are leakage-free.

A clean pipeline with K-Means often beats a leaky pipeline with GMM.

Data leakage is the #1 cause of ML project failure. A pipeline prevents it automatically.

How Do You Prevent Data Leakage Automatically?

Standard unsupervised pipeline:

1. `SimpleImputer(strategy='median')`
2. `StandardScaler()`
3. `PCA(n_components=10)`
4. `KMeans(n_clusters=4)`

Each step transforms data for the next. PCA *after* scaling (never before): PCA requires centered, scaled data.

For time series: replace `cross_val_score` with `TimeSeriesSplit`.

Deployment: serialize with `joblib.dump()` — one file contains the entire pipeline.

Read the chart:

- Flow: Data → Impute → Scale → PCA → Cluster
- Each box = one pipeline step
- The pipeline wraps all steps into one leakage-free object

As you learned in L32, the pipeline guarantees correct ordering and prevents leakage. One `joblib.dump()` saves everything.

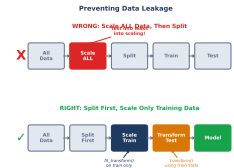
Where Exactly Does Leakage Happen in Unsupervised Learning?

Common mistake: fit StandardScaler on *all* data, then split into train/test, then cluster.
The scaler "knows" the mean and standard deviation of the test data — information has leaked.

Correct approach: fit scaler on training data only; transform test data using training statistics.

UL-specific leakage:

- Fitting PCA on all data before train/test split — principal components are contaminated
- Computing silhouette on data used for fitting — score is inflated



Read the chart:

- Leaky pipeline (red) fits scaler on all data
- Clean pipeline (green) fits on train only
- The gap can be 5–15% on financial data

If your clustering looks too good in a notebook, check for leakage. The honest score is always lower.

Why Can't You Shuffle Financial Data for Cross-Validation?

Standard K-fold CV shuffles data randomly — past and future mix. For financial time series, future data must **never** leak into training.

TimeSeriesSplit (expanding-window CV):

- Fold 1: train months 1–12, test 13–15
- Fold 2: train months 1–15, test 16–18
- Training expands; test slides forward

```
tscv = TimeSeriesSplit(n_splits=5)
```

Use when clustering stock returns or building regime-detection pipelines on financial data.

Read the chart:

- Training (navy) always precedes test (amber) in time
- The training window expands each fold
- No shuffling — chronological order preserved

For financial data, always validate forward in time. Shuffled CV pretends you can see the future — that is lookahead bias.

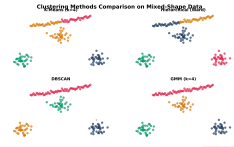
Do Different Methods Agree on the Same Data?

Apply K-Means, hierarchical (Ward), DBSCAN, and GMM to the **same** 2D dataset:

- **K-Means:** spherical, fast, forces all points into a cluster
- **Ward:** similar to K-Means; provides dendrogram
- **DBSCAN:** follows density, identifies noise, sensitive to ϵ
- **GMM:** soft assignments, elliptical, slower

No single method wins on all data. Choose based on cluster shape, noise, and analytical goal.

UL in NLP: LDA for long docs, BERTopic for short text — topic modeling remains unsupervised even in the transformer era.



Read the chart:

- Four panels: K-Means, Ward, DBSCAN, GMM
- Where clusters are clear, all methods agree
- Disagreement = ambiguous boundaries worth investigating

If all four methods give the same answer, you have strong clusters. If they disagree, investigate why before choosing.

When Is Unsupervised Learning the Wrong Tool?

Do NOT use unsupervised learning when:

- **You have labels.** Use supervised learning. Clustering labeled data wastes information.
- **The task has a clear target variable.** Prediction $\Rightarrow$ supervised. Exploration $\Rightarrow$ unsupervised.
- **Clusters are not meaningful.** K-Means will *always* find K clusters, even in random data. Validate with silhouette scores and domain knowledge.
- **The result is not actionable.** Ask: "What decision does this clustering inform?" Clustering for its own sake has no value.
- **Data is too small.** With < 30 points, any clustering is noise. Use domain expertise instead.

Knowing when NOT to use a tool is as important as knowing how to use it. Unsupervised learning is for exploration, not prediction.

Clustering Formula Reference Sheet

K-Means

Inertia: $J = \sum_{k=1}^{K} \sum_{x \in C_k} \|x - \mu_k\|^2$

Centroid update: $\mu_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$

Distance Metrics

Euclidean: $d = \sqrt{\sum (x_i - y_i)^2}$

Manhattan: $d = \sum |x_i - y_i|$

Cosine: $\cos\theta = \frac{x \cdot y}{\|x\| \, \|y\|}$

Silhouette Score

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

$a(i)$ = avg. intra-cluster dist.

$b(i)$ = min avg. inter-cluster dist.

Range: $[-1, +1]$

Ward Linkage

Merge to minimize increase in total within-cluster variance.

DBSCAN

Core point: $\geq$ MinPts within ϵ

MinPts $\geq 2d$ (rule of thumb)

GMM Mixture

$P(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x | \mu_k, \Sigma_k)$

$\sum_k \pi_k = 1$

EM Algorithm

E: compute $P(\text{cluster}|x)$

M: update μ_k, Σ_k, π_k

BIC

BIC $= -2 \ln L + k \ln n$

Lower = better.

Print this slide. You will need it for the exam.

Dimensionality Reduction Formula Reference Sheet

PCA
Covariance: $C = \frac{1}{n-1} X^\top X$
Eigen: $Cv_i = \lambda_i v_i$
Cum. var.: $\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$
Projection: $Z = X \cdot V_k$
(V_k = first k eigenvectors)

Kaiser Criterion
Keep PCs with $\lambda_i > 1$

Reconstruction Error
$\|X - XV_k V_k^\top\|^2$

Loadings
v_{ij} = weight of feature j in PC i
Sign = direction of contribution

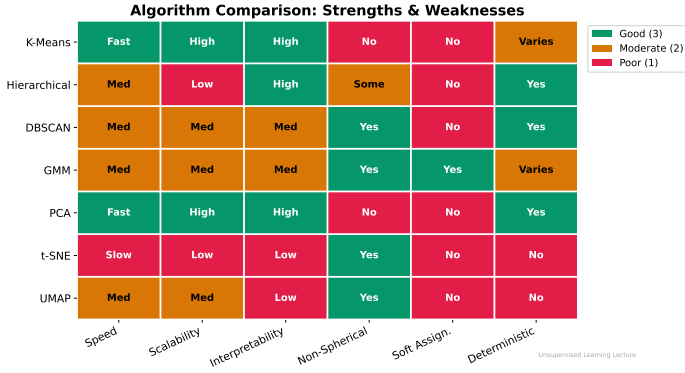
t-SNE
Perplexity $\approx$ neighbors (5–50)
$O(n^2)$ time

UMAP
`n_neighbors` = context
$O(n \log n)$ time
15–25% higher trustworthiness

Autoencoder
Nonlinear $f : \mathbb{R}^p \to \mathbb{R}^k$
$\sim$100$\times$ slower than PCA

Print this slide alongside the clustering formula sheet. Together they cover every formula in this lecture.

Which Algorithm Scores Best on Each Criterion?



Read the chart: Green = strong, amber = moderate, red = weak. No method is green across all criteria. DBSCAN handles noise and shape; GMM provides probabilities — they complement each other.

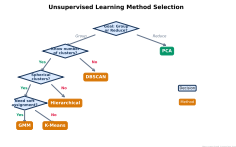
***Hierarchical does not require K in advance but must be cut to produce clusters. No method is universally best.**

Which Unsupervised Method Should You Use?

Decision procedure:

1. "Cluster or reduce dimensions?"
2. **Cluster:** Know K? Yes $\rightarrow$ K-Means/GMM. No $\rightarrow$ DBSCAN/hierarchical.
3. **Reduce:** Interpretable? Yes $\rightarrow$ PCA. No $\rightarrow$ UMAP/t-SNE.
4. **Shape:** Spherical? $\rightarrow$ K-Means. Non-spherical? $\rightarrow$ DBSCAN/GMM.
5. **Noise/Size:** Outliers $\rightarrow$ HDBSCAN. >10K pts $\rightarrow$ K-Means/UMAP.

Always validate with silhouette, scree, or domain knowledge.



Read the chart:

- Diamonds = questions; rectangles = recommendations
- Start at "What is your goal?" and follow arrows
- Branch on shape, noise, K known, and dataset size

The best method is the one whose assumptions match your data. Start with the flowchart, then validate empirically.

Where Can You Learn More About Unsupervised Learning?

Textbooks (free online):

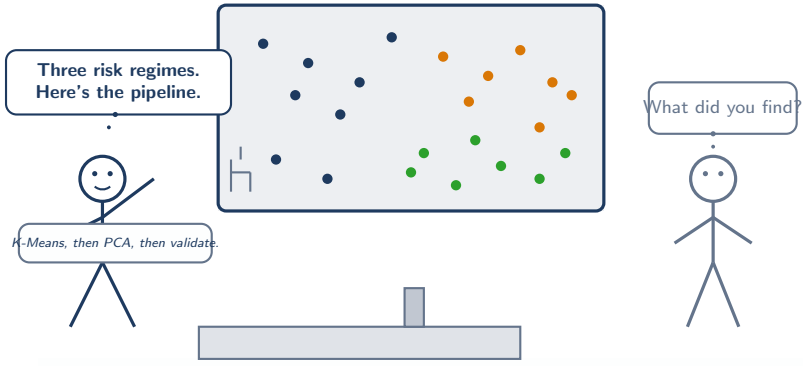
1. James et al., *ISLR*, Ch. 12 — BSc level
2. Hastie et al., *ESL*, Ch. 14 — graduate level
3. Géron, *Hands-On ML*, Part I — practical, sklearn-focused

Finance: Lopez de Prado, *Advances in Financial ML* (2018), Ch. 16 (HRP) and Ch. 7 (PCA).

This course: L29 (K-Means), L30 (Hierarchical), L31 (PCA), L32 (Pipeline). Deep Learning L33–L36 covers autoencoders.

Online: scikit-learn.org/stable/modules/clustering.html
scikit-learn.org/stable/modules/decomposition.html

The best way to learn UL is to explore real data. Pick a dataset, cluster it, reduce it, and ask: "Does this grouping make sense?"



Unsupervised Learning: From K-Means to Gaussian Mixtures