COM SCI M148, Lecture 9:
Unsupervised Learning (Part 2)

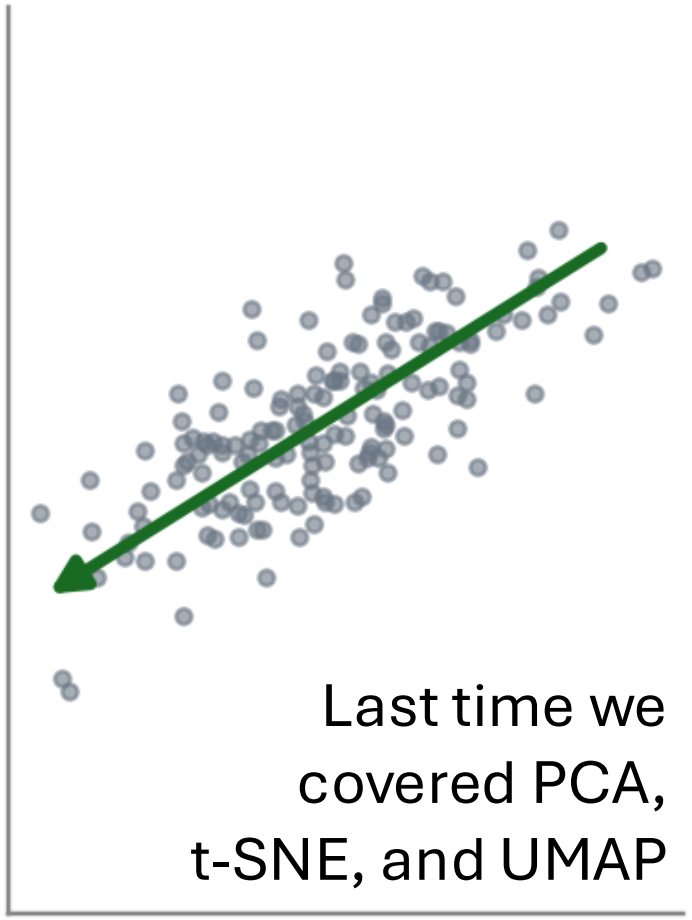
Prof. Nathan Szymanski
nszymanski@seas.ucla.edu
Department of Materials Science

UCLA

Samueli
School of Engineering

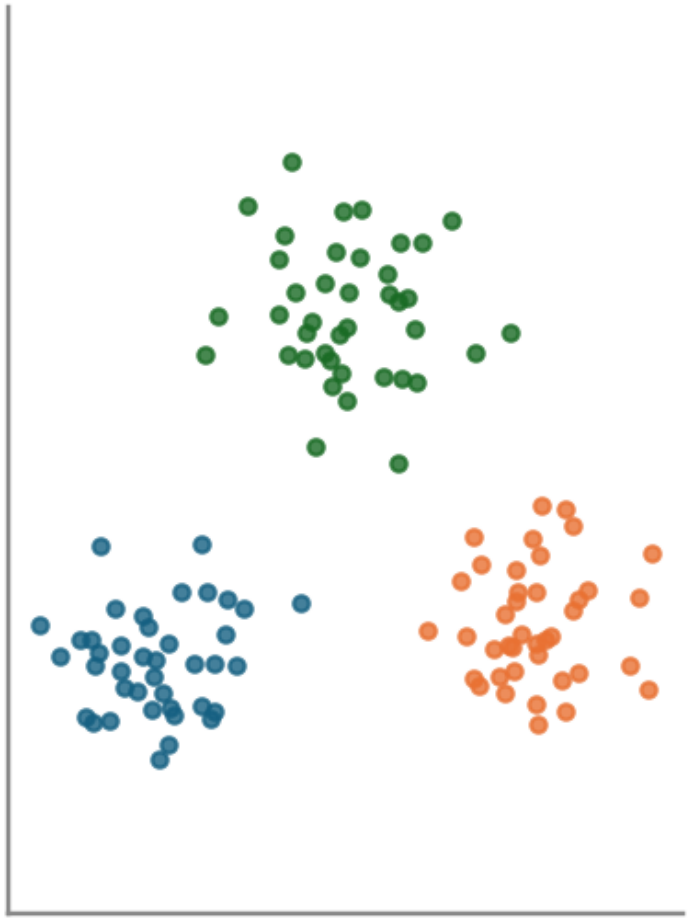
Three unsupervised learning problems

Dimensionality reduction



Last time we covered PCA, t-SNE, and UMAP

Clustering



Anomaly detection

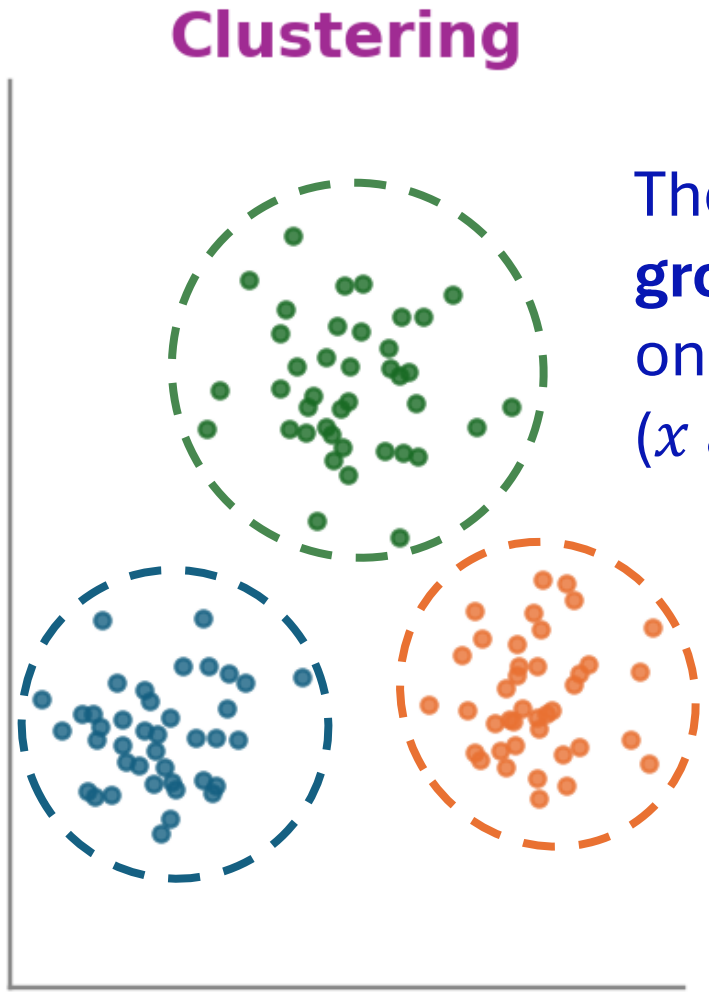


Topic for today: Clustering

Automatically **group** data points so that points in a group are **similar** to each other, and **different** from other groups

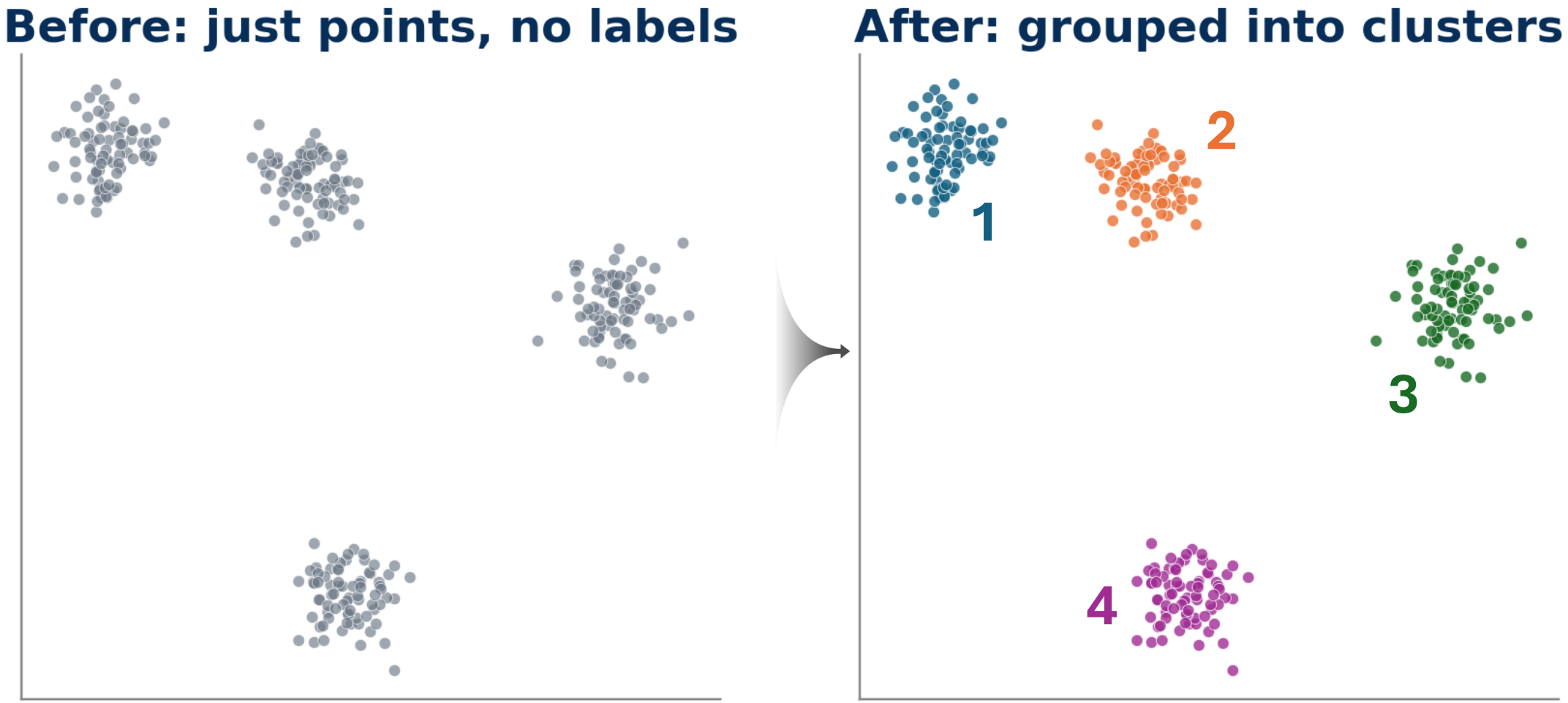
Keep in mind: We still don't have labels!

So…This is *not* supervised learning



The idea is to ***learn* these groups of labels** based on trends in the features (x and y)

Clustering assigns data into groups



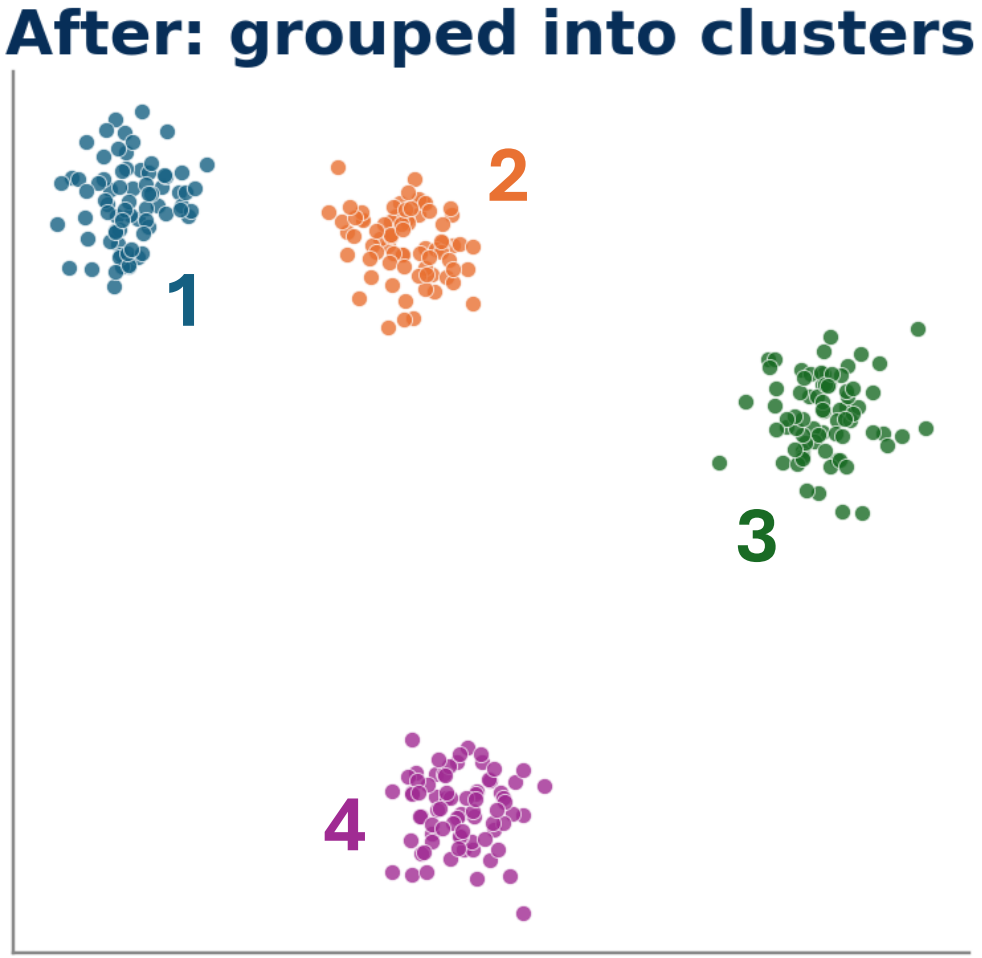
Unlike PCA, we are not transforming any coordinates – just assigning labels

Clustering *vs.* Classification (not the same!)

Clustering ≠ Classification

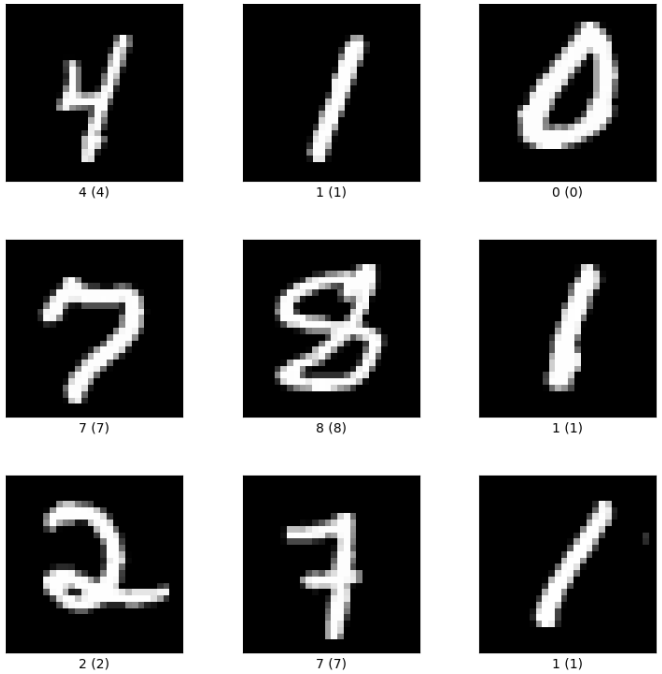
Discovers groups without being provided labels. The assigned "group labels" may or may not be meaningful

Learns to map features onto labels, which are provided in the training data

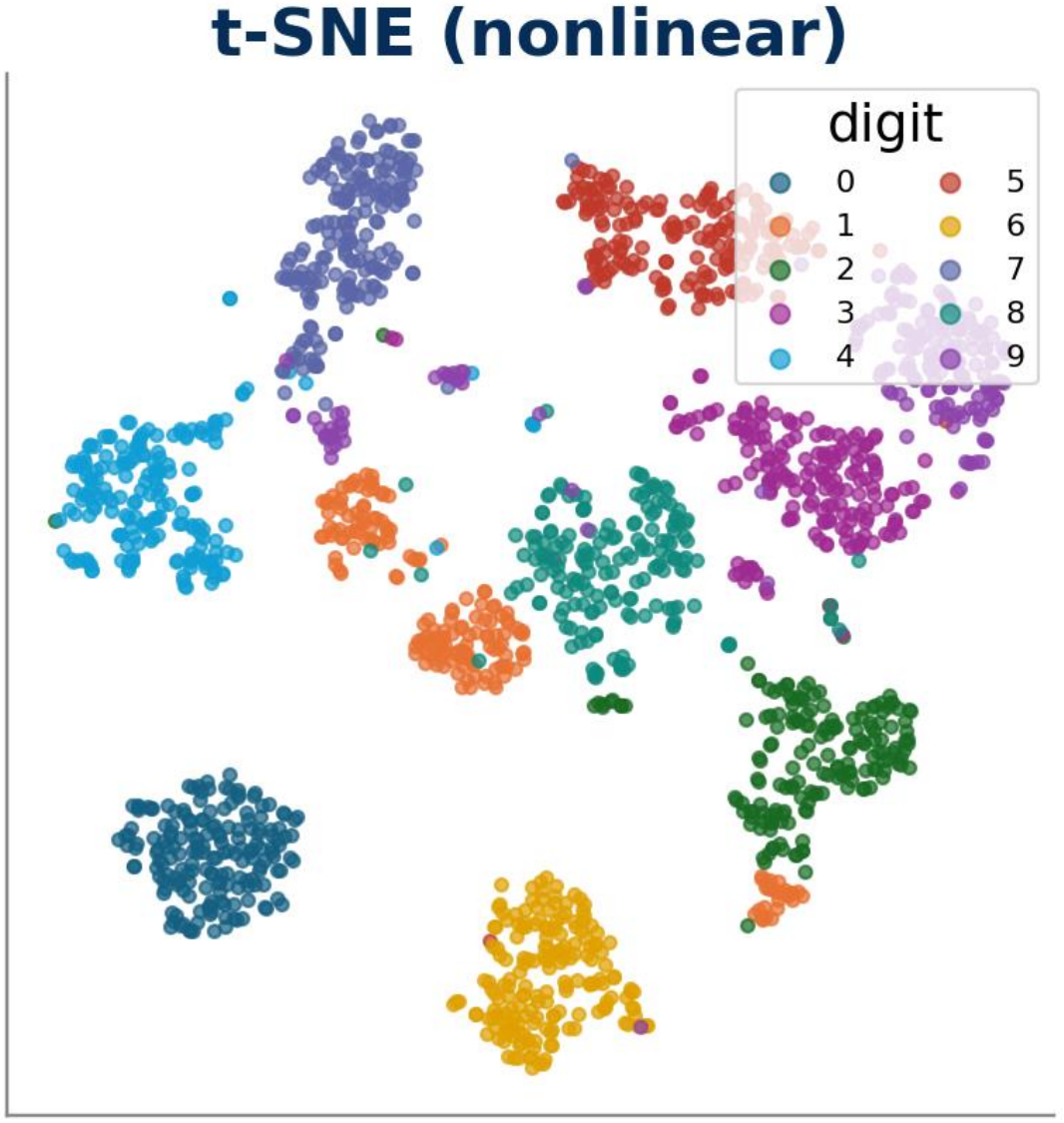


Unlike PCA, we are not transforming any coordinates – just assigning labels

Clustering examples from last time



MNIST:
A dataset of handwritten digits



Here, the data seems to group naturally along its true labels

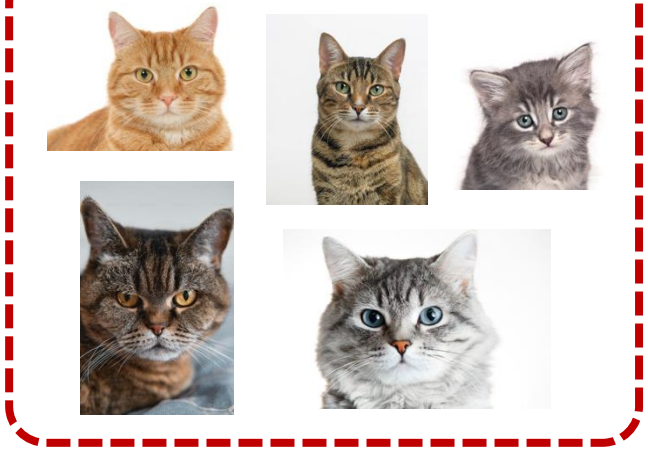
Clustering examples from last time

These images have **millions of pixels** and would be difficult to separate using hand crafted rules…



Organize them by similarity in a high-dimensional space, or after dimensional reduction, without having any knowledge of the true labels (cat/dog)





K-Means Clustering

Partition the data into **k groups**

K-means clustering: Step-by-step

1. Pick K cluster centers (centroids) where K is a hyperparameter

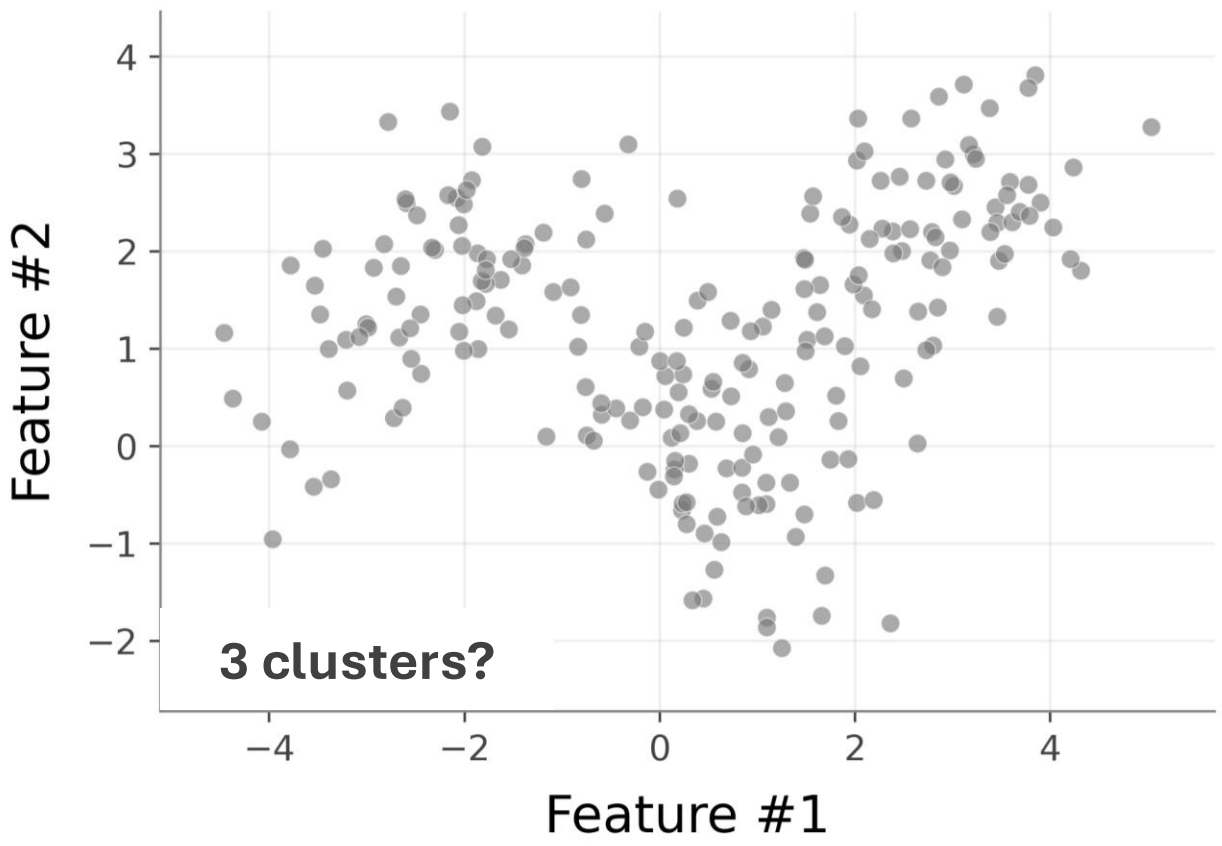


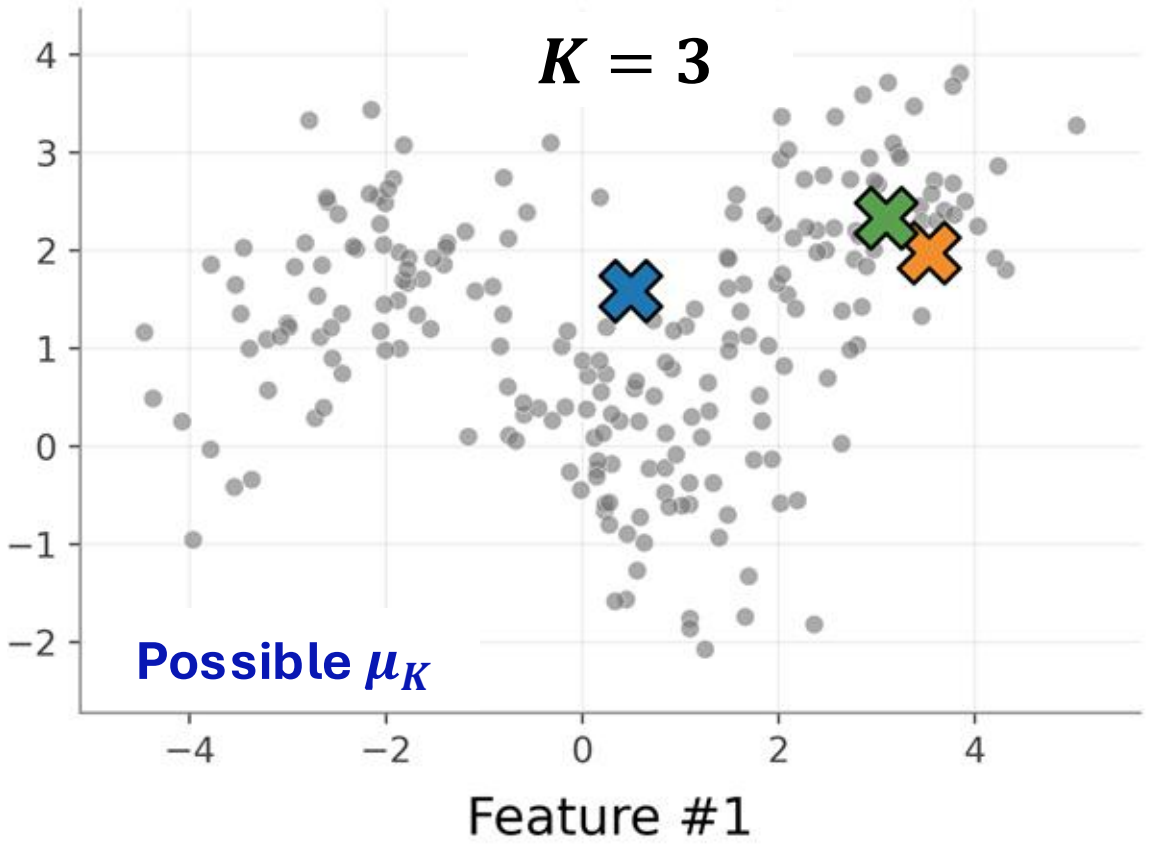
- Typically, K is chosen based on the number of clusters you expect
- If you have no idea, try several values and see how the results vary (we'll come back to this)
- **K centroids (μ_K) are randomly initialized within the range of the data**

μ_K = centroid coordinates

K-means clustering: Step-by-step

1. Pick K cluster centers (centroids) where K is a hyperparameter



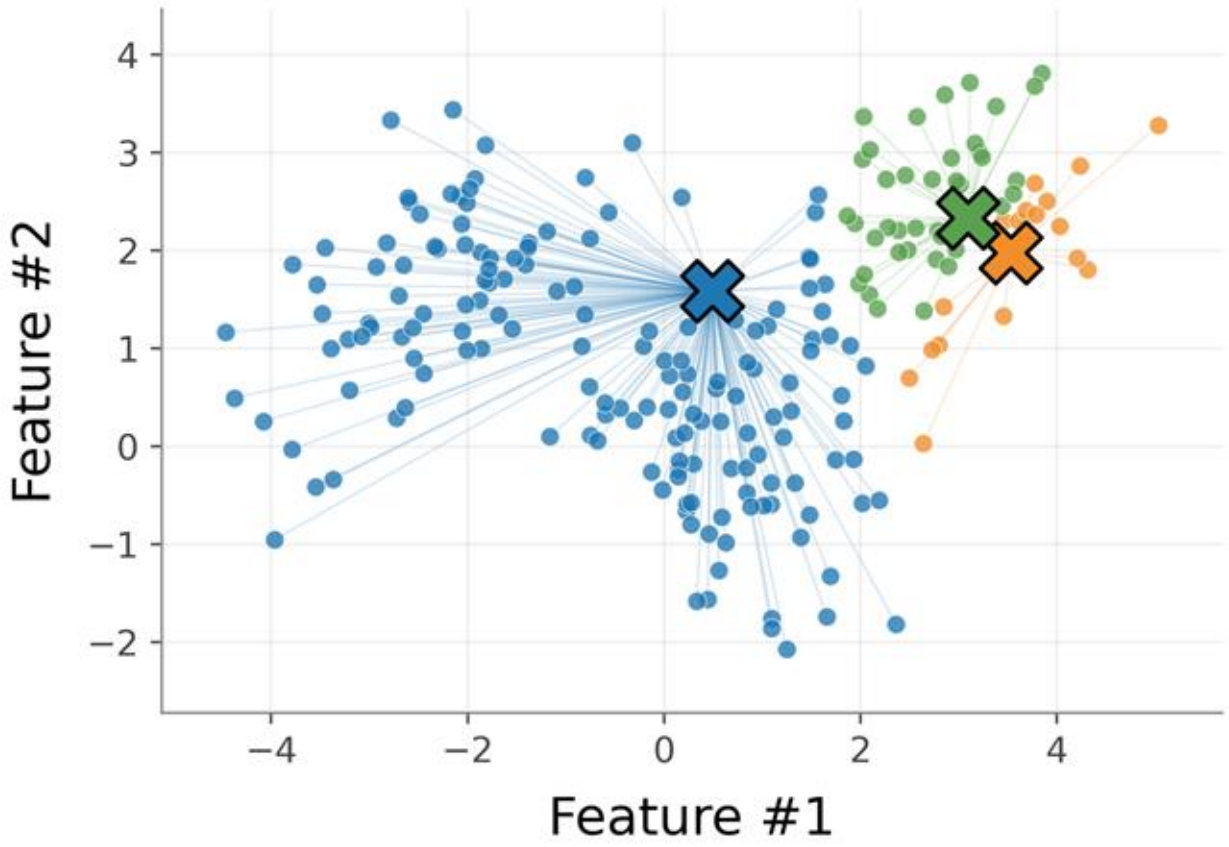


K-means clustering: Step-by-step

2. Assign each datapoint to its nearest centroid

$$c_i = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmin}} \|x_i - \mu_k\|^2$$

We assign the cluster label (c_i) according to which centroid is "nearest" – *i.e.*, which has the **smallest Euclidean distance (L2 norm)**

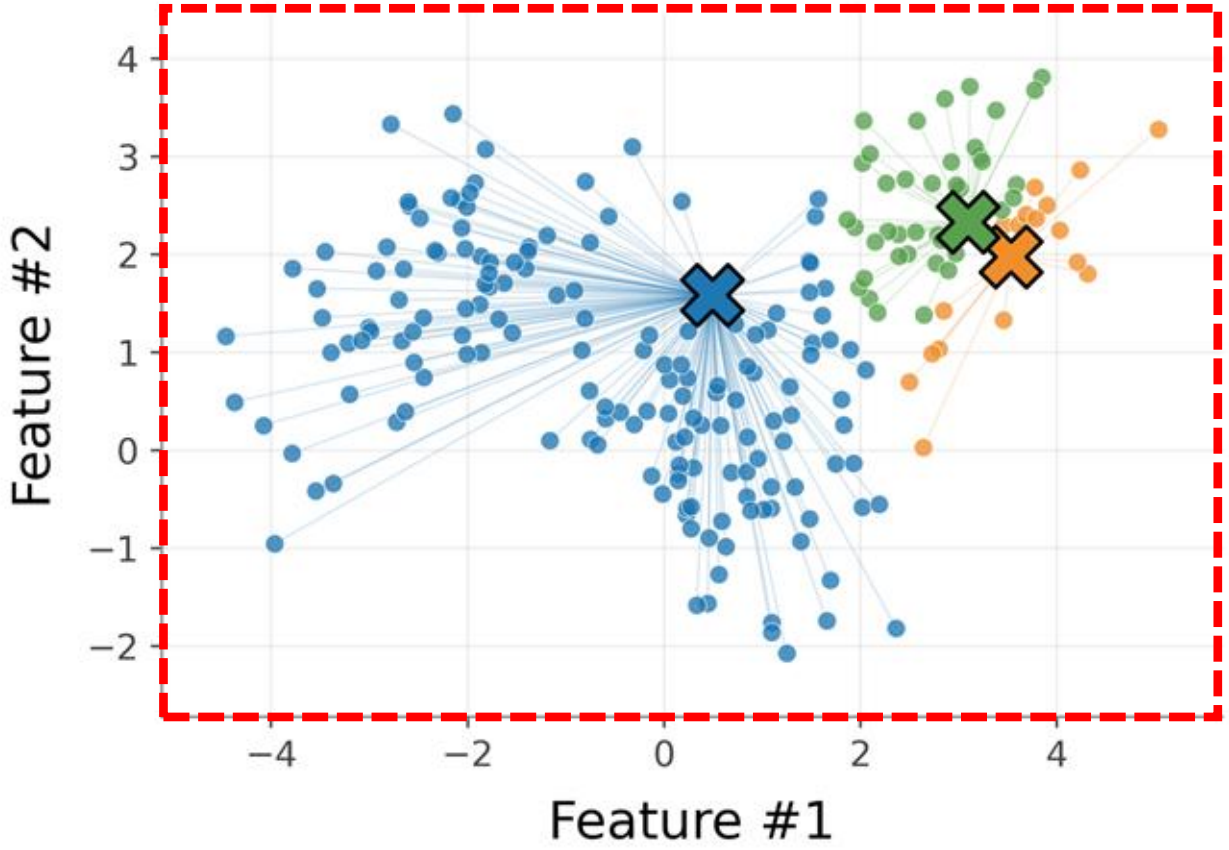


K-means clustering: Step-by-step

2. Assign each datapoint to its nearest centroid

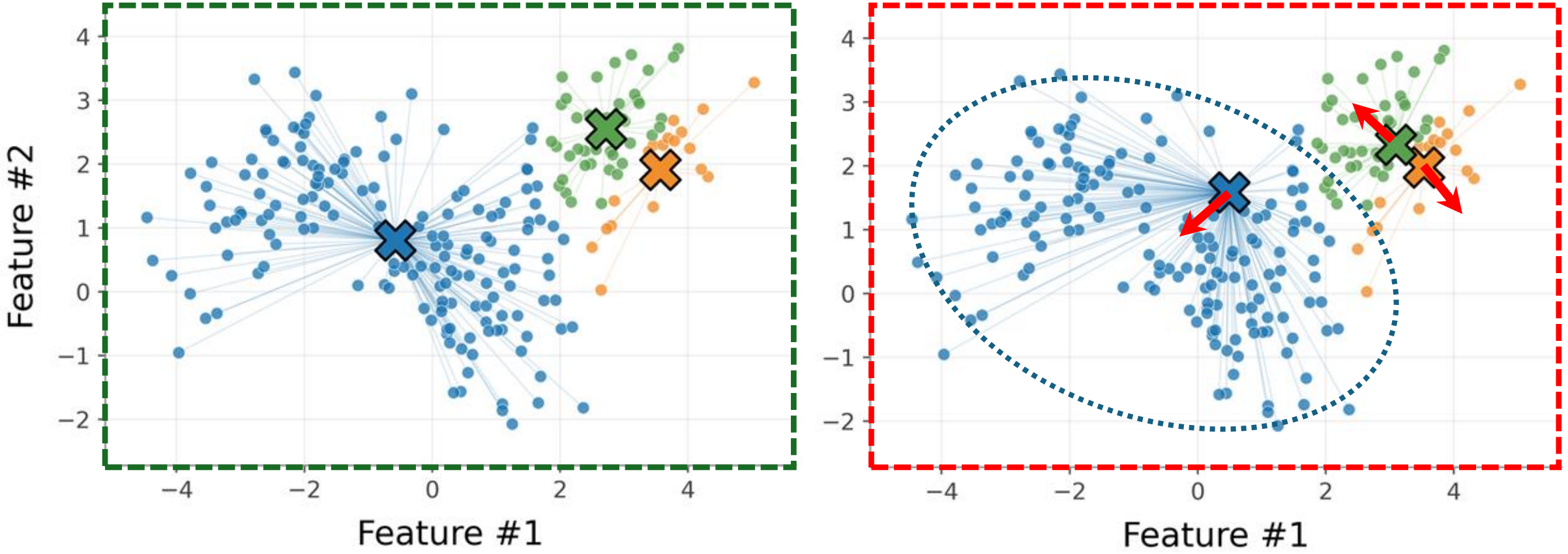
Does this cluster assignment look ideal? Probably not...

Let's adjust the positions of the centroids to better represent their connected points



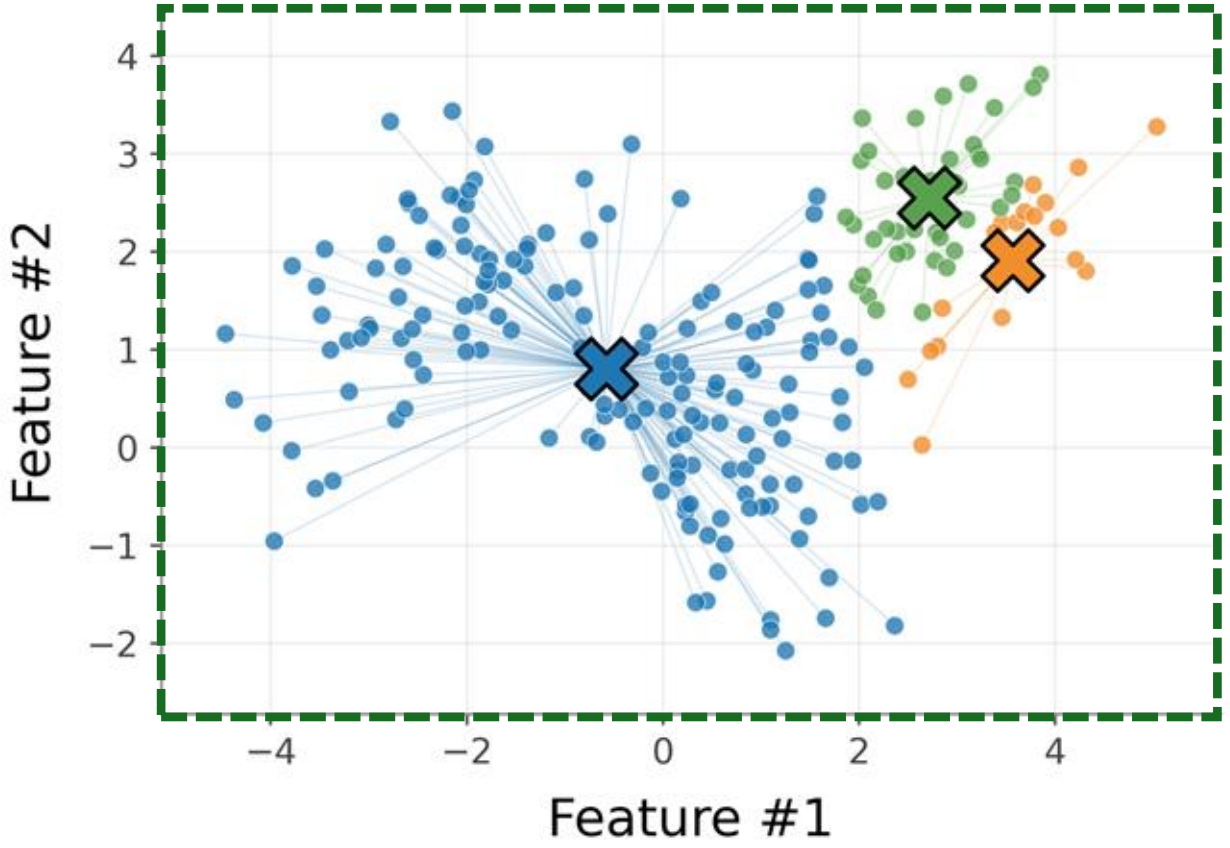
K-means clustering: Step-by-step

3. Recompute each centroid as the mean of all points assigned to it



K-means clustering: Step-by-step

3. Recompute each centroid as the mean of all points assigned to it

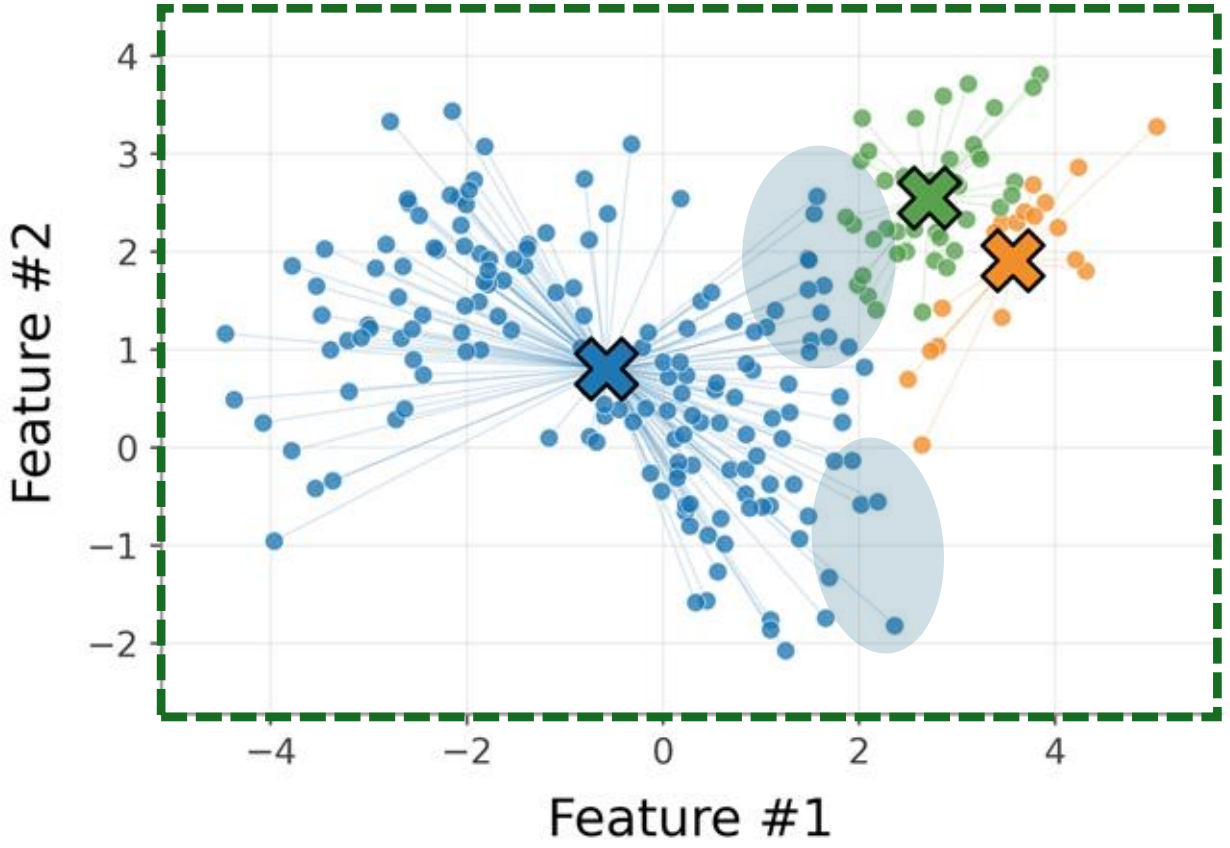


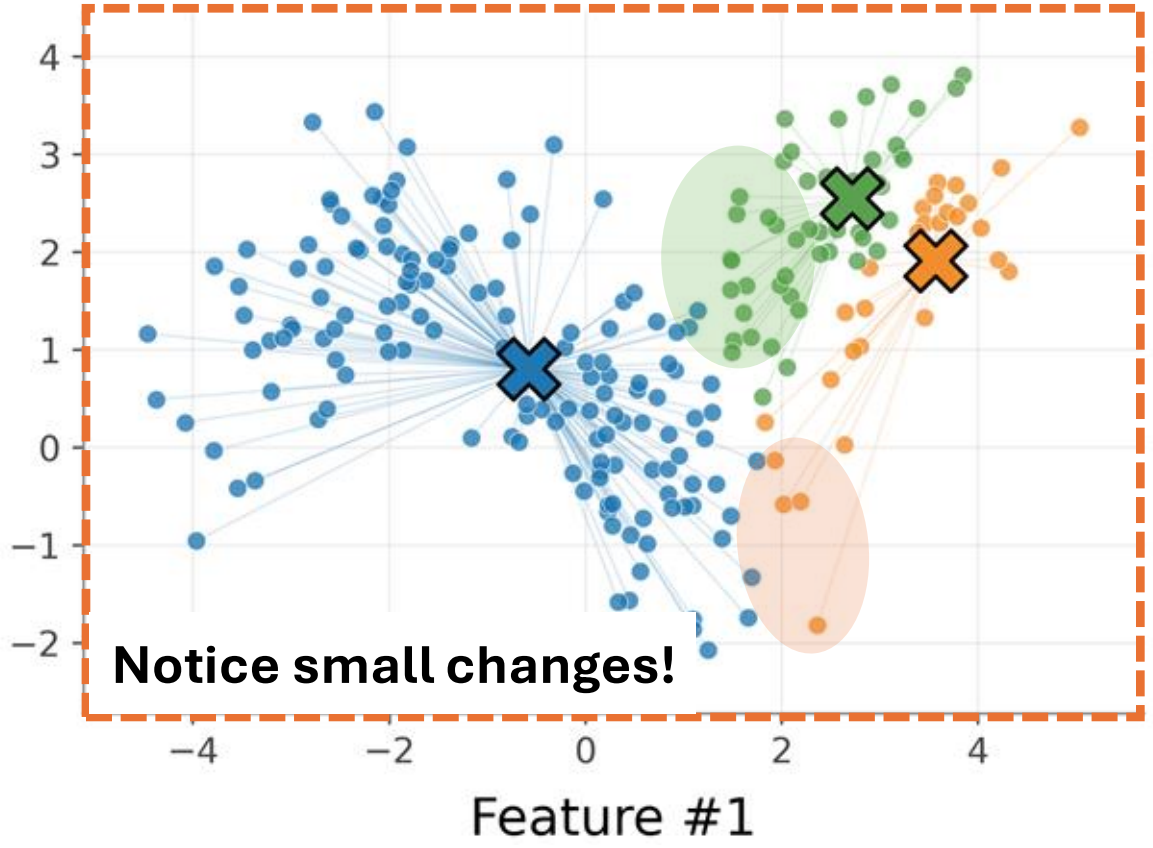
$$\boldsymbol{\mu'_k} = \frac{1}{n} \sum_{x_i \in c_k} \boldsymbol{x_i}$$

Sum over the points around each centroid ($x_i \in c_k$) and divide by the total number of points in that cluster (n)

K-means clustering: Step-by-step

2. Assign each datapoint to its nearest centroid*...Again!*





K-means clustering: Step-by-step

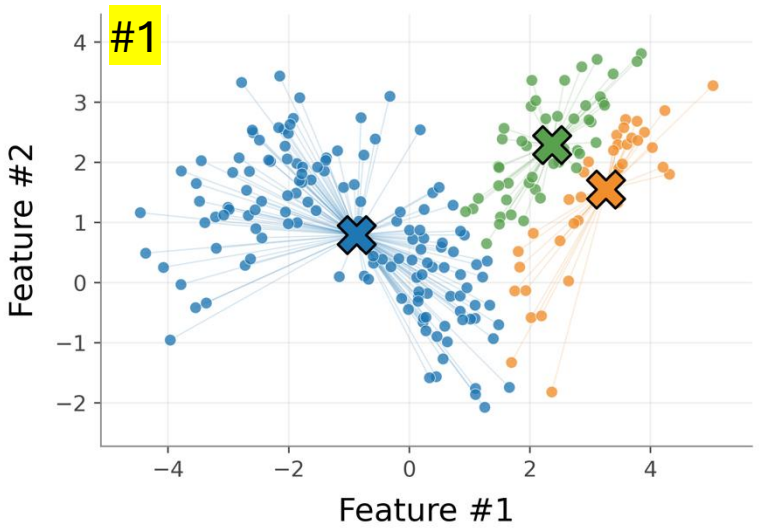
1. Pick K cluster centers (centroids) where K is a hyperparameter

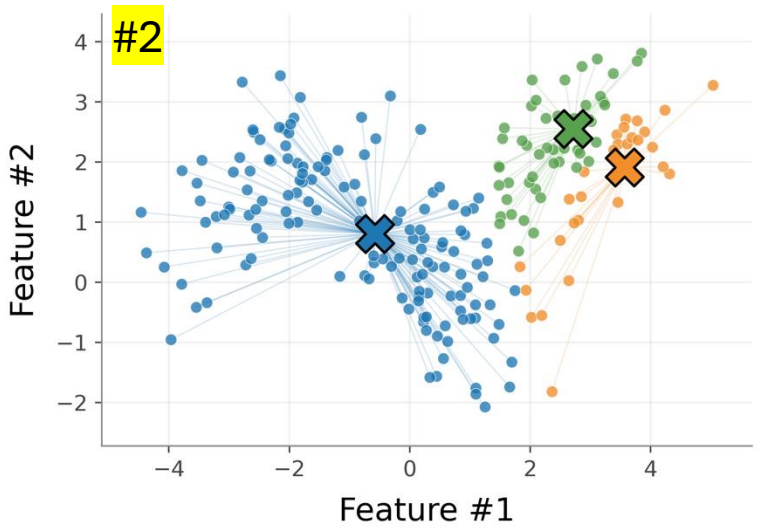
2. Assign each datapoint to its nearest centroid

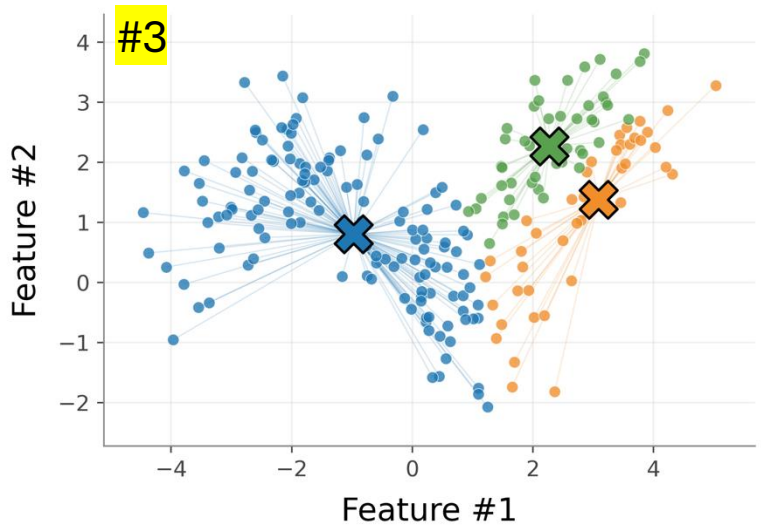
3. Recompute each centroid as the mean of all points assigned to it

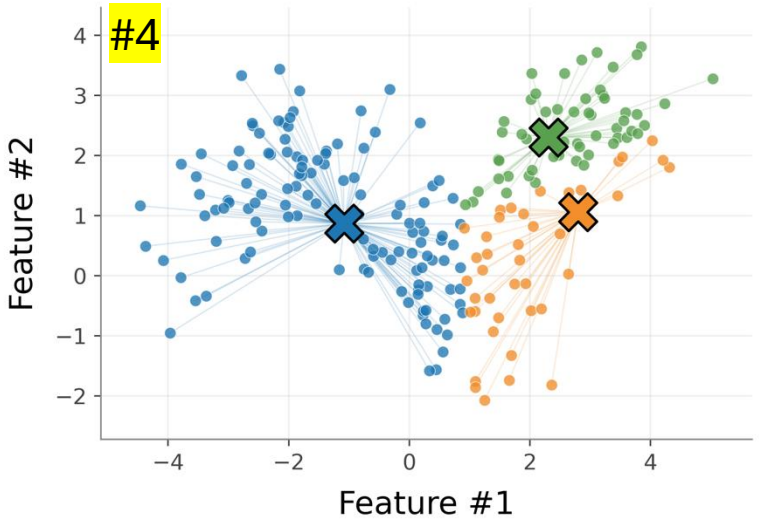
Repeat until convergence is reached
(when assignments no longer change)

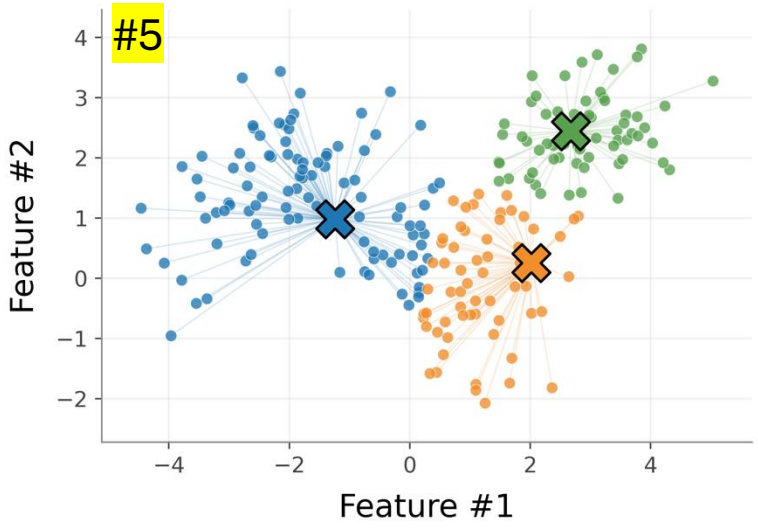
Iterate until assignments no longer change

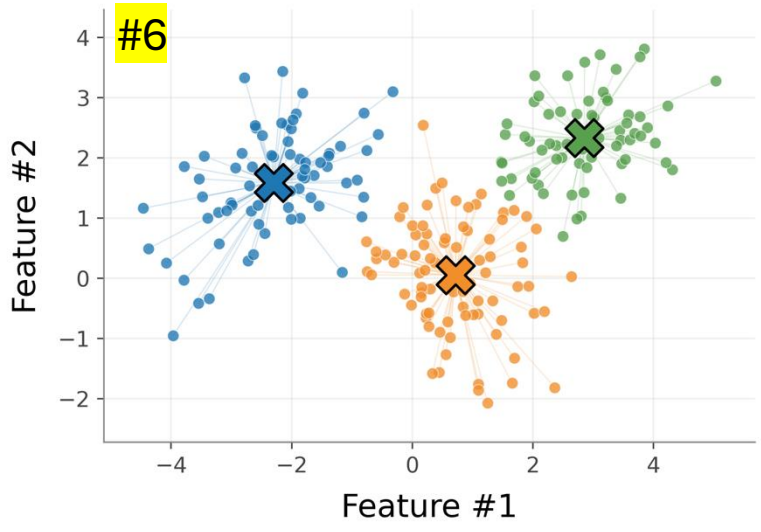












Iterate until assignments no longer change

At each iteration, we check:

Did any assignments change?

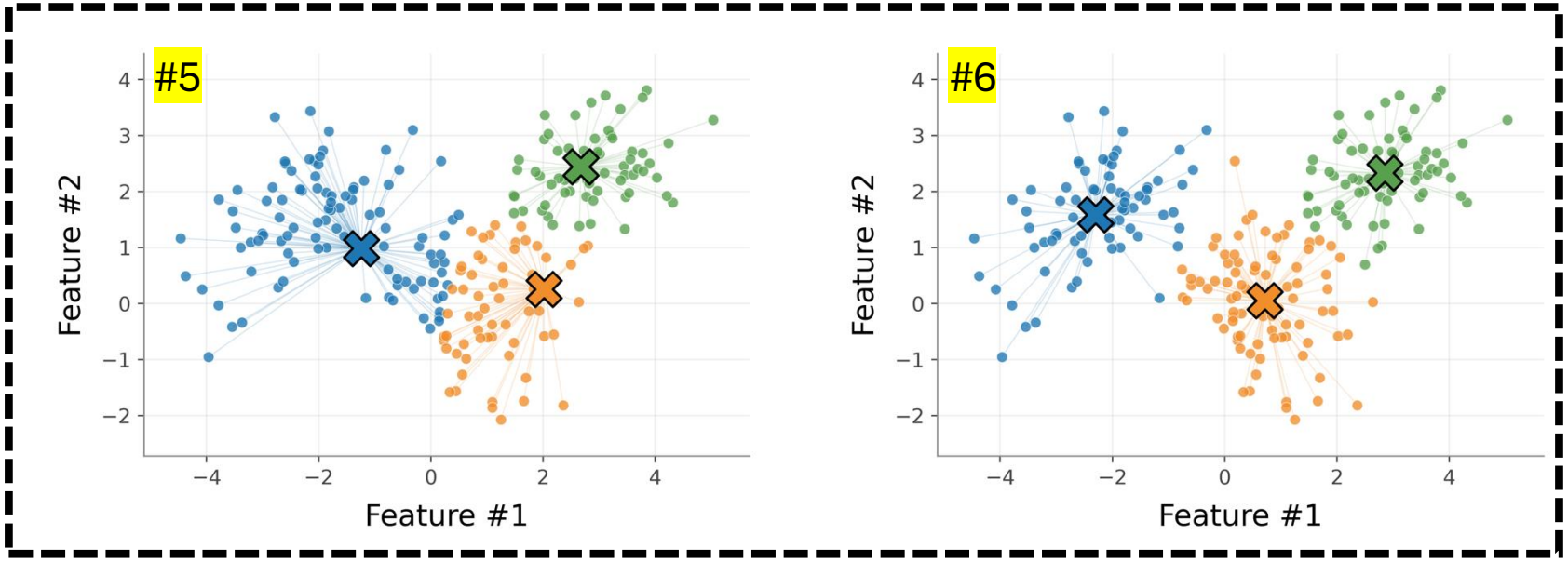
Yes

Repeat the process

No… $c_i^t = c_i^{t+1}$

All done!

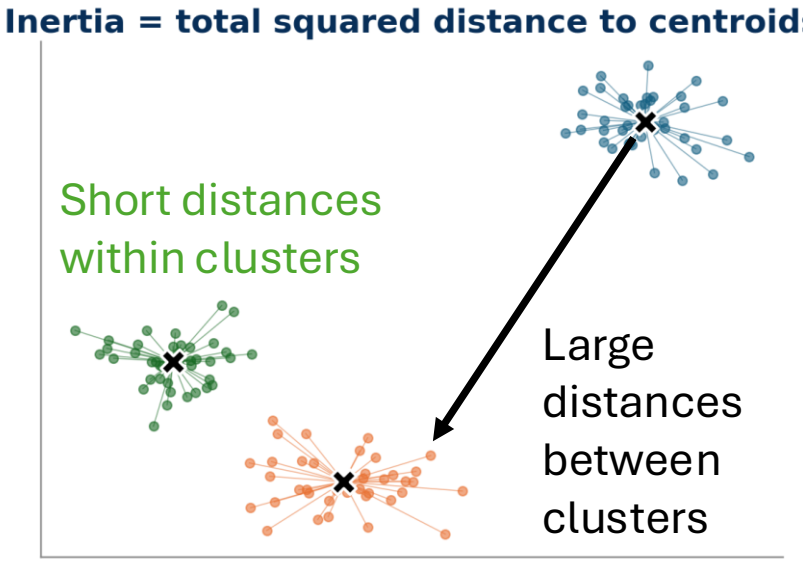
Or if you have lots of data and are impatient, you can stop at a threshold: sufficiently small movements in all the centroids



Questions?

K-means clustering minimizes inertia

This iterative process, known as **Lloyd's algorithm** is equivalent to minimizing the "within-cluster" sum of squared distances, also called the **inertia** J:



K-means tries to make in-cluster distances as small as possible!

$$\textbf{Minimize } J = \sum_{i=0}^{n} \left\| x_i - \mu_{c_i} \right\|^2$$

- We sum over every data point (x_i)

  ...And compute its distance to...

- μ_{c_i} = centroid of the cluster c_i is assigned to

Each step lowers it, so **K-means always converges...**

But *not* necessarily to the global minimum

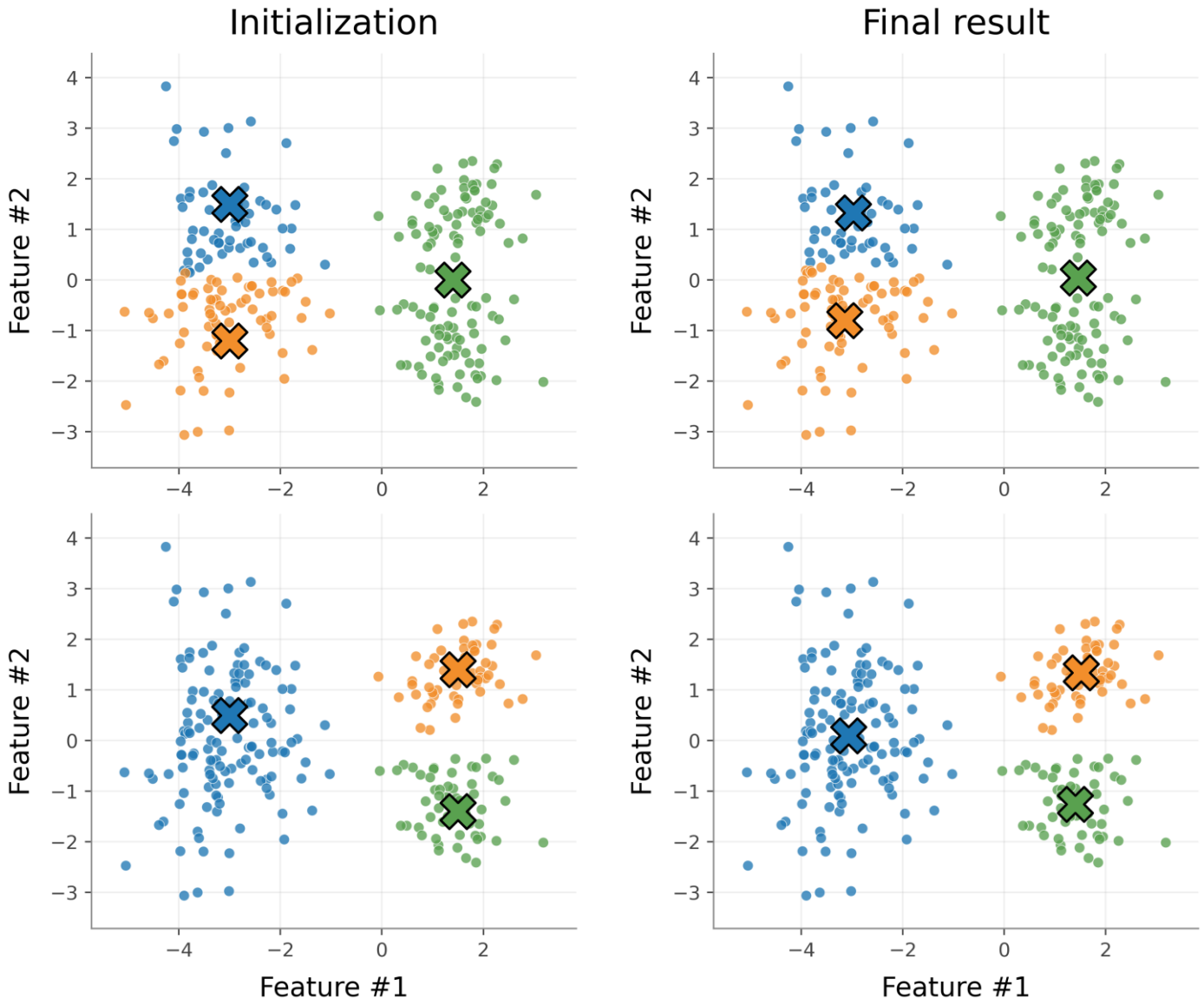
Initialization matters!

Final within-cluster sum of squared distances:

$$J = 423.52$$

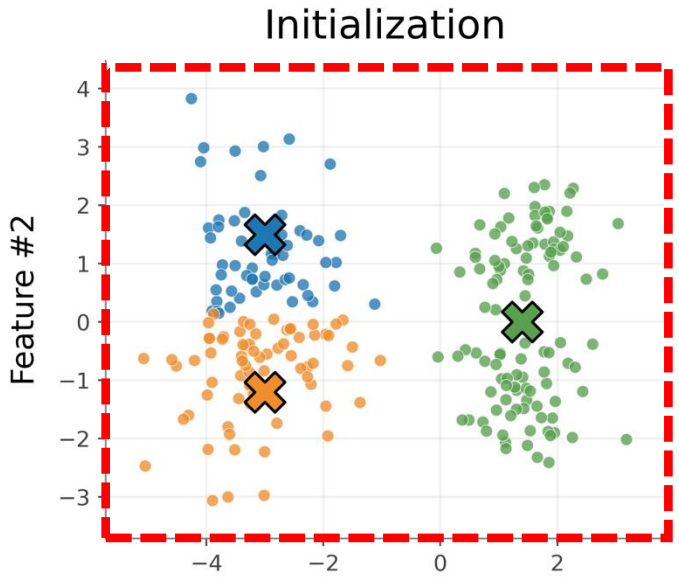
Final within-cluster sum of squared distances:

$$J = 382.54$$



Initialization matters!

Option 1:
Try several different random initializations, run Lloyd's algorithm multiple times, and keep the best result



We'd like to avoid "bad" initialization of the centroids

Option 2: K-Means++

1. Choose the first centroid at random

2. Compute the distance to every point

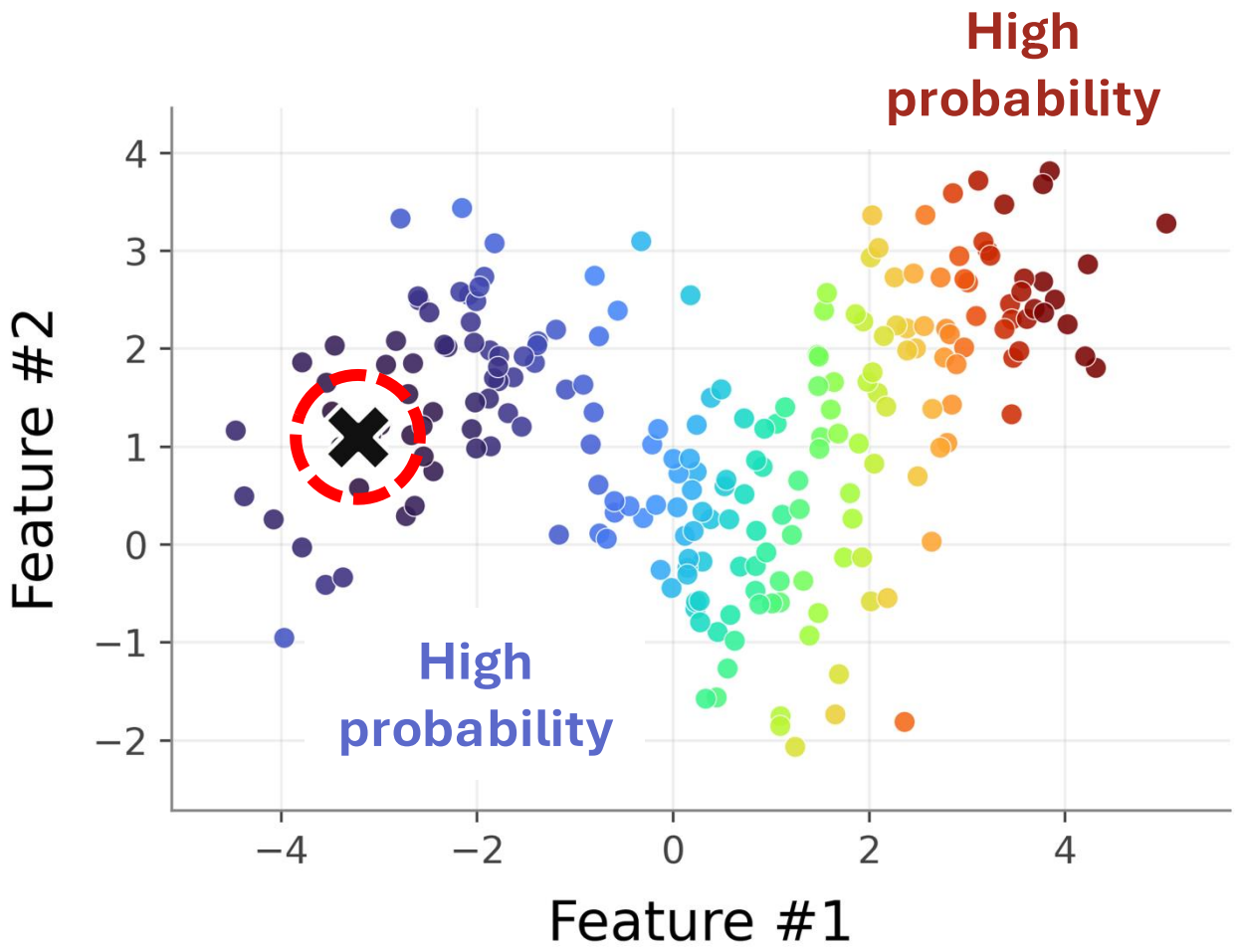
$$D(x_i) = \min_k \|x_i - \mu_k\|^2$$

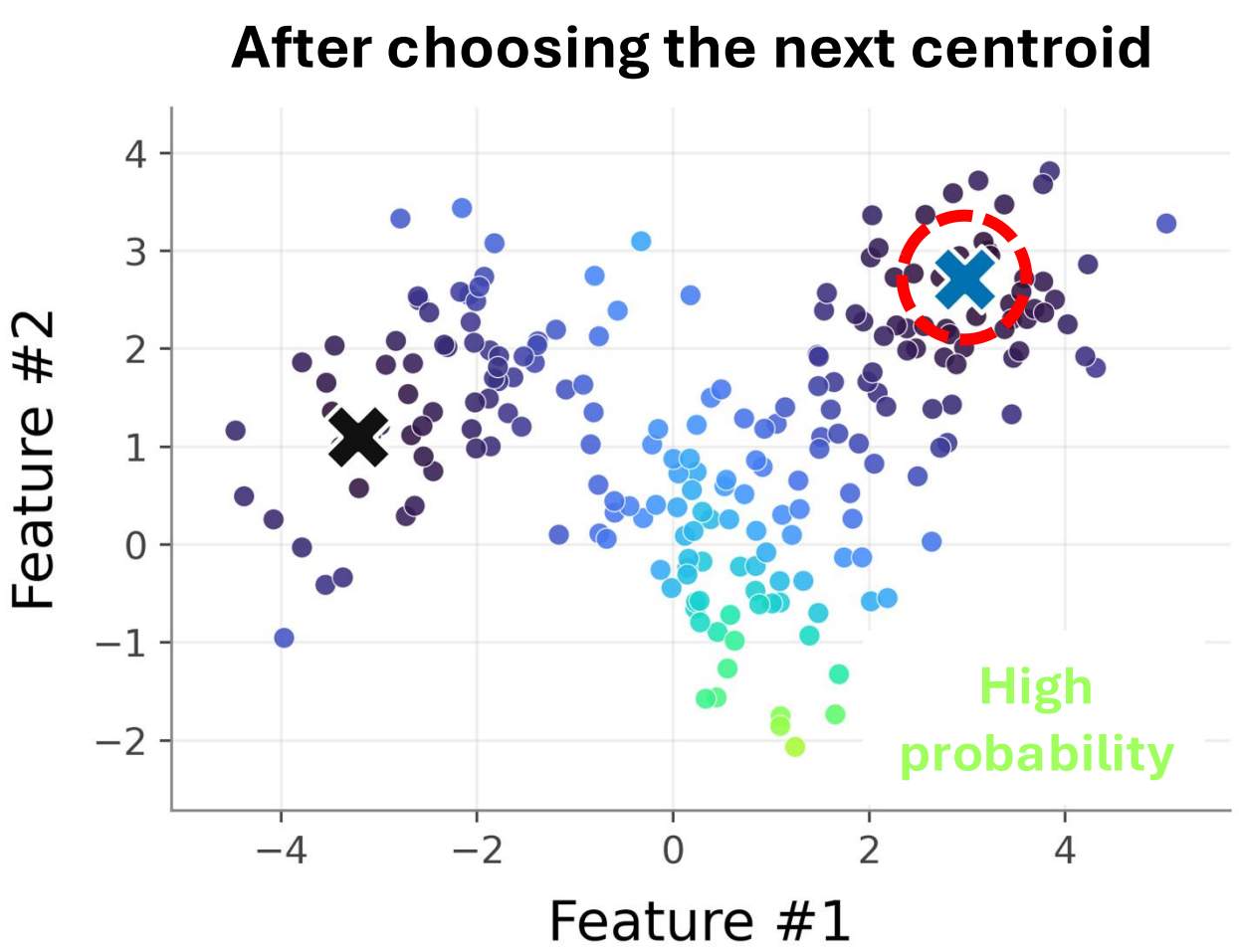
3. Choose the next centroid randomly but with probability ∝ square distance

$$P(x_i) = \frac{D(x_i)^2}{\sum_j D(x_j)^2}$$

Points far away from existing centroid(s) are more likely to be chosen!

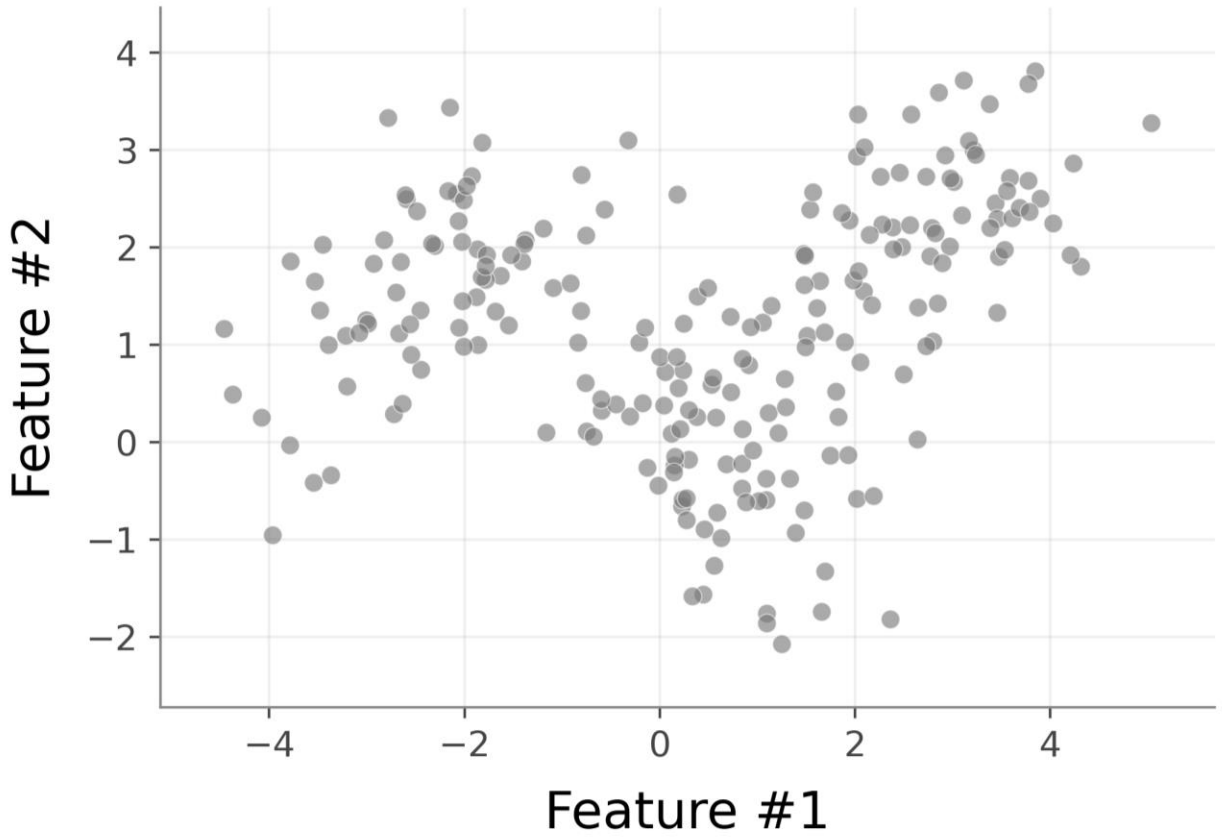
K-means++


High probability
High probability
Feature #2
Feature #1
4
3
2
1
0
−1
−2
−4
−2
0
2
4


After choosing the next centroid
High probability
Feature #2
Feature #1
4
3
2
1
0
−1
−2
−4
−2
0
2
4

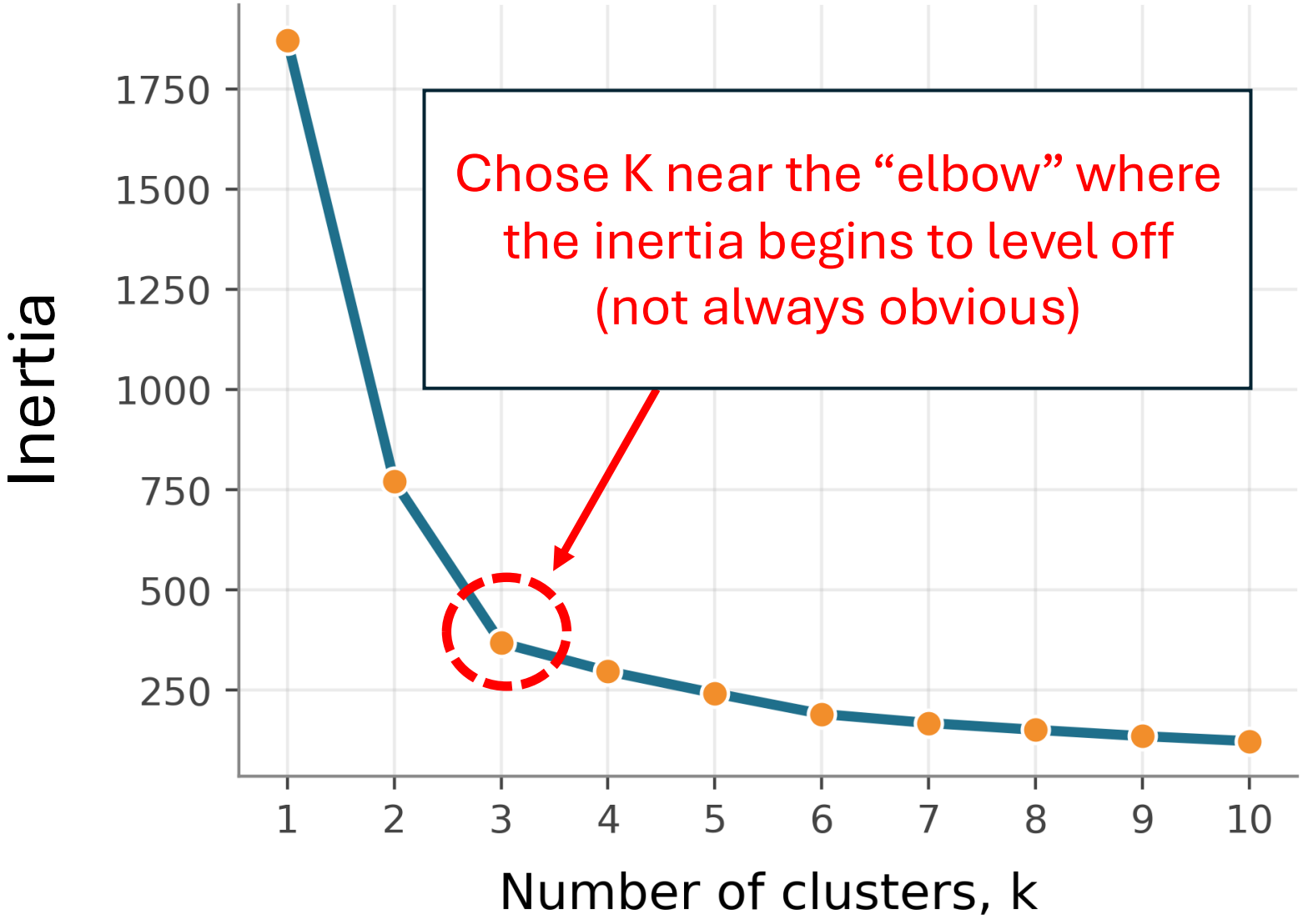
How to choose the number of clusters (K)

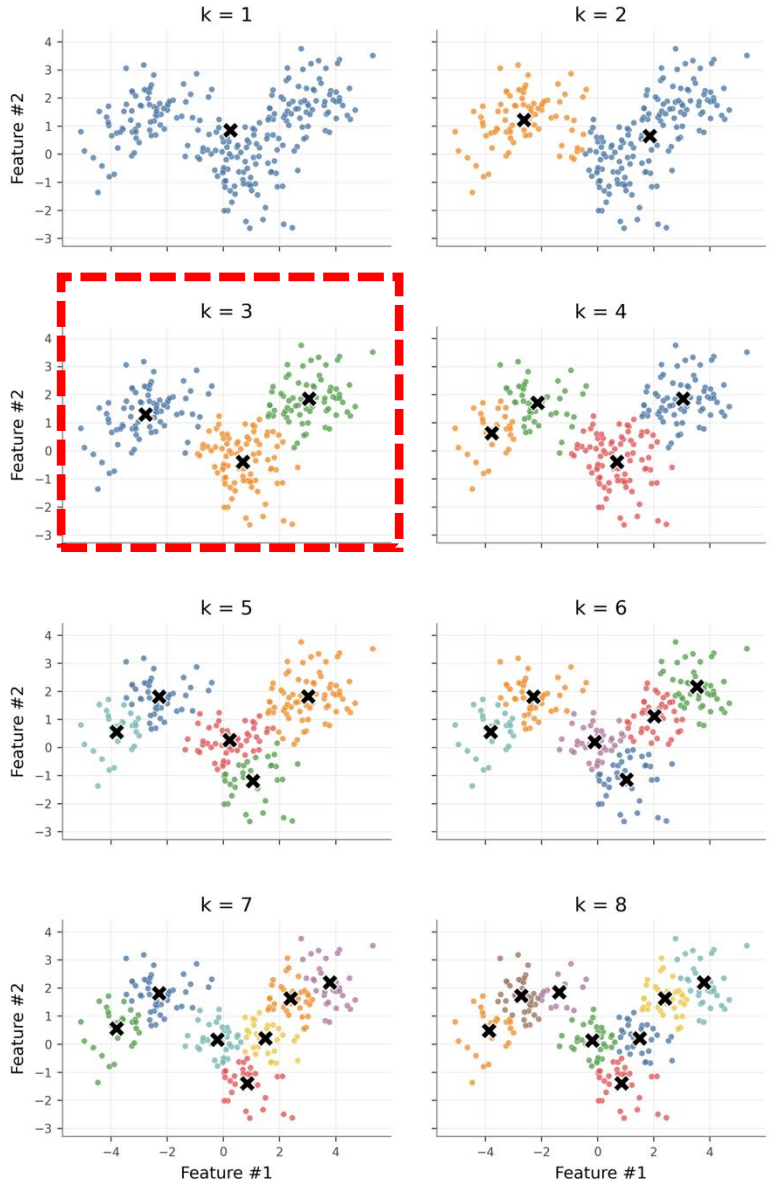
1. Pick K cluster centers (*i.e.*, centroids) where K is a hyperparameter



- Typically, K is chosen based on the number of clusters you expect
- If you have no idea, try several values and see how the results vary (discussed on next slide)
- **K centroids (μ_K) are randomly initialized (or using K-means++) from the dataset itself**

Option #1: The elbow method



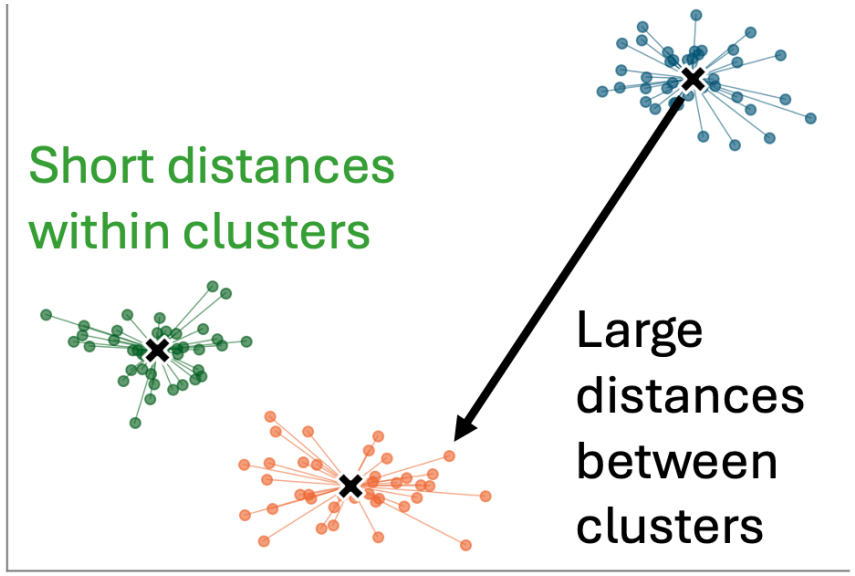


Option #2: The silhouette score

For each point i we compute:

Cohesion:
a_i: the average **distance from that point to all other points in its own cluster**

Separation:
b_i: the average distance from the point to all points in the **nearest other cluster**



Remember, we want something like this…
Strong cohesion (short distances in-cluster)
Large separation between clusters

Option #2: The silhouette score

For each point i we compute:

Cohesion:
a_i: the average **distance from that point to all other points in its own cluster**

Separation:
b_i: the average distance from the point to all points in the **nearest other cluster**

The silhouette score is:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Normalize by the larger distance (usually separation)

$s_i \approx 1 \rightarrow b_i$ **large,** a_i **small** → the point is much closer to its own cluster than any other cluster **(what we want!)**

$s_i \approx 0 \rightarrow b_i$ **and** a_i **comparable** → the point is in between clusters, with ambiguous assignment **(what we do *not* want!)**

Option #2: The silhouette score

For each point i we compute:

Cohesion:
a_i: the average **distance from that point to all other points in its own cluster**

Separation:
b_i: the average distance from the point to all points in the **nearest other cluster**

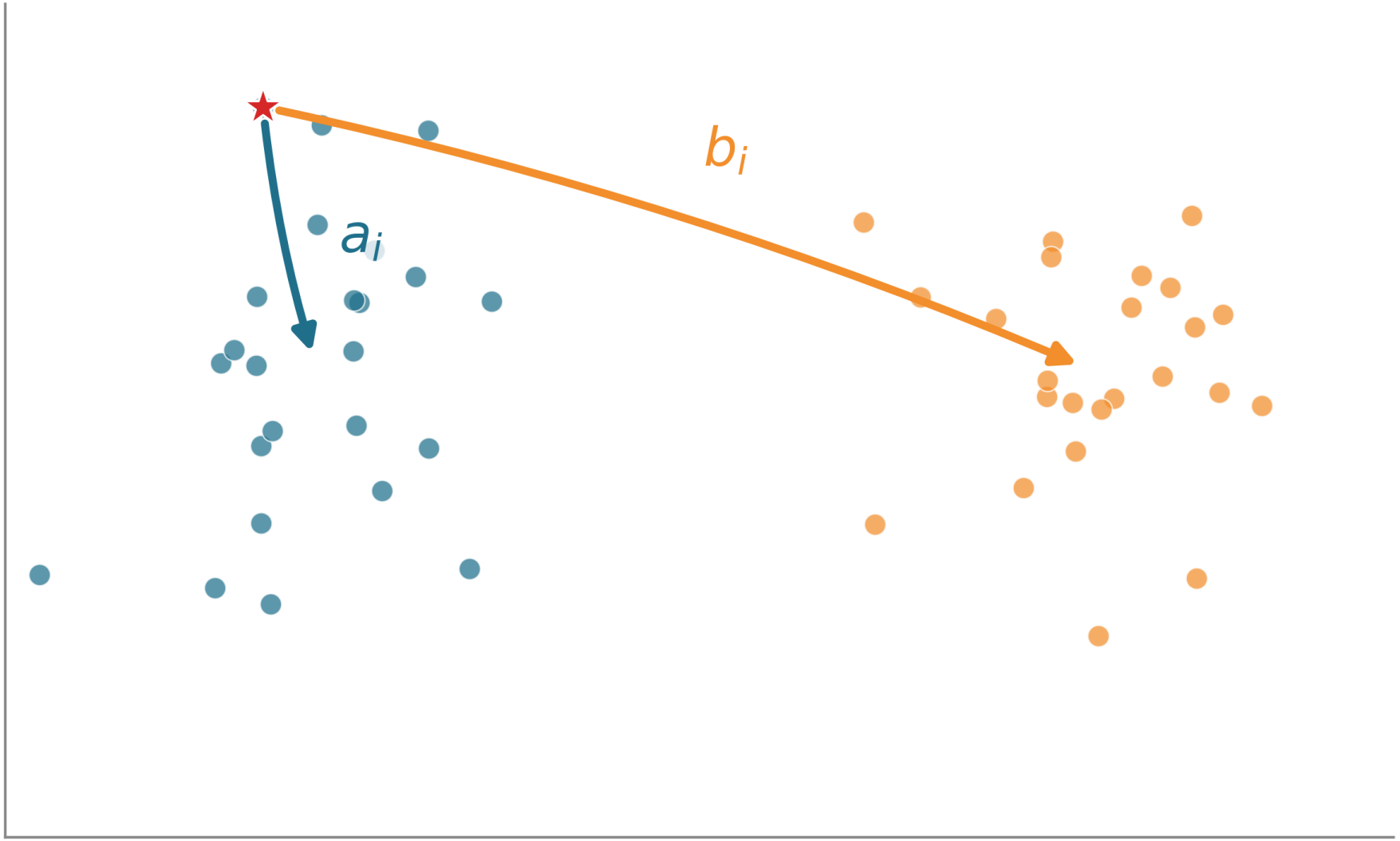
The silhouette score is:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Normalize by the larger distance (usually separation)

$s_i < 0 \rightarrow b_i$ **is smaller than** $a_i \rightarrow$ the point is closer to another cluster than its own...
Bad assignment!

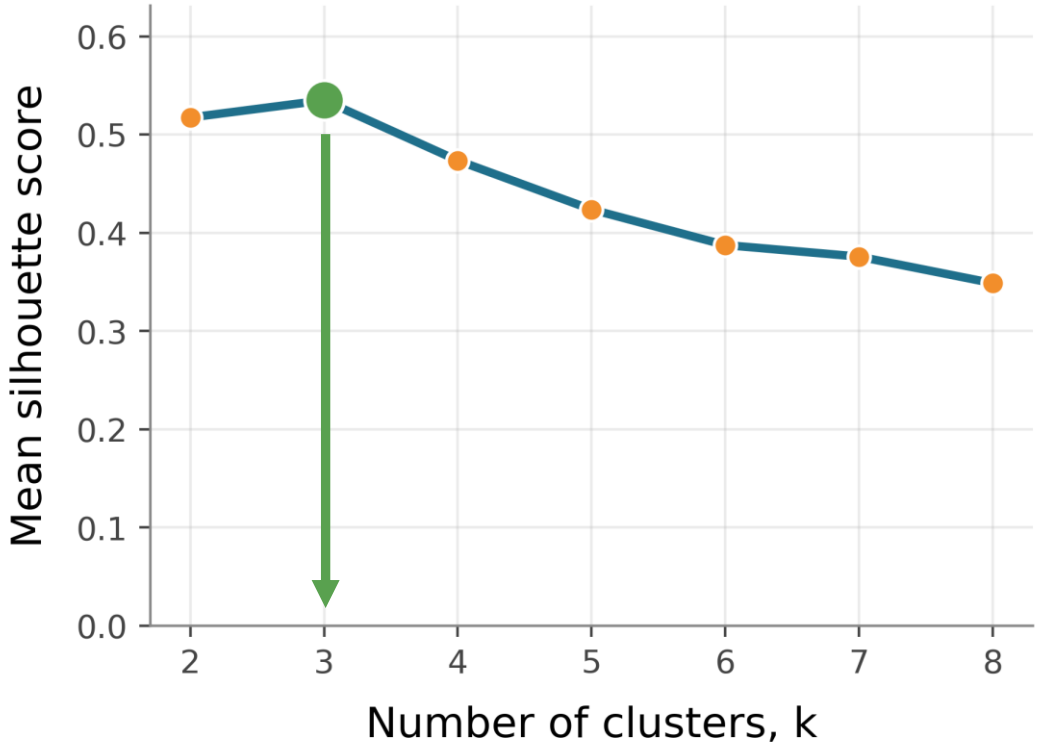
How to use this to choose K…



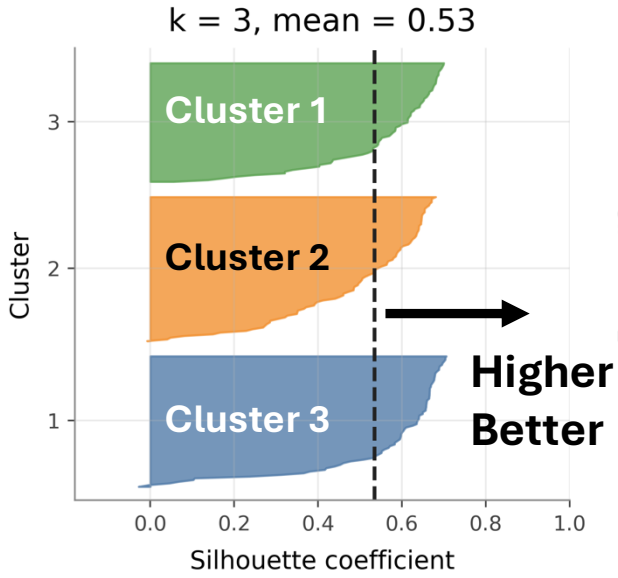
We want b_i to be large (separation) and a_i to be small (cohesion)

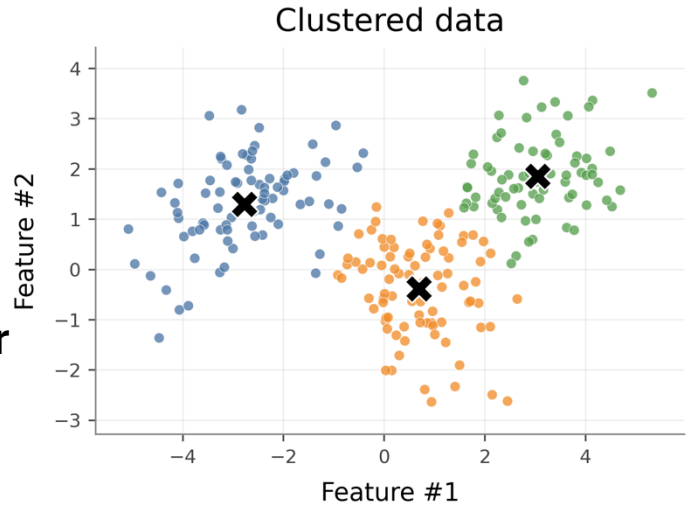
Find the score for all points in each cluster, plot their distributions, and compute the average

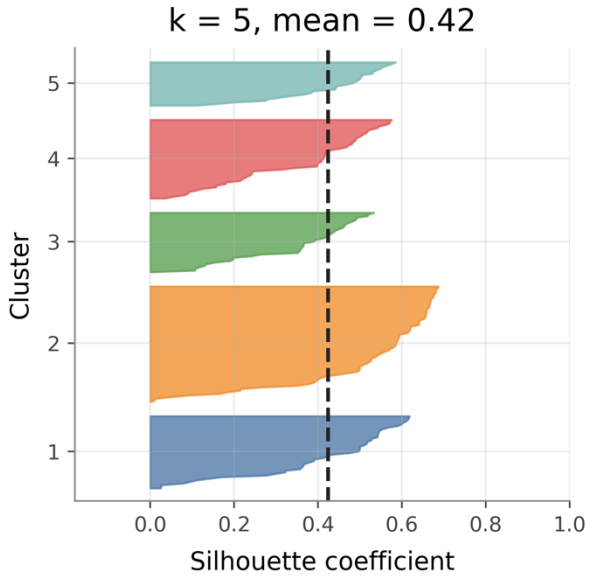
How to use this to choose K…

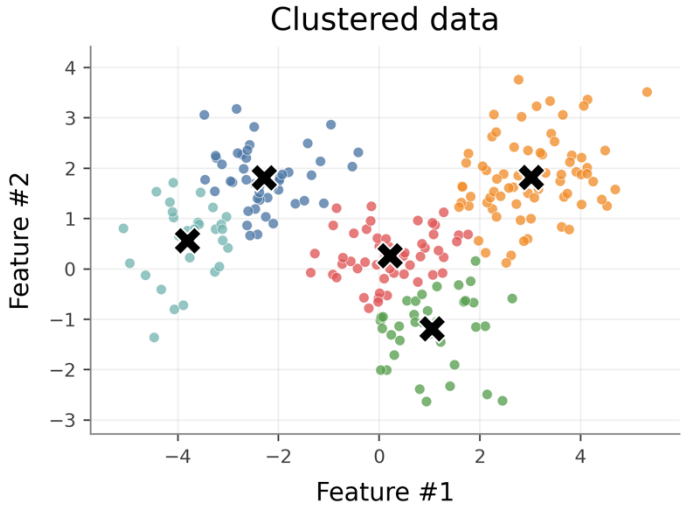


Choose K where the average silhouette score is maximal
(this is usually more reliable than the elbow method)

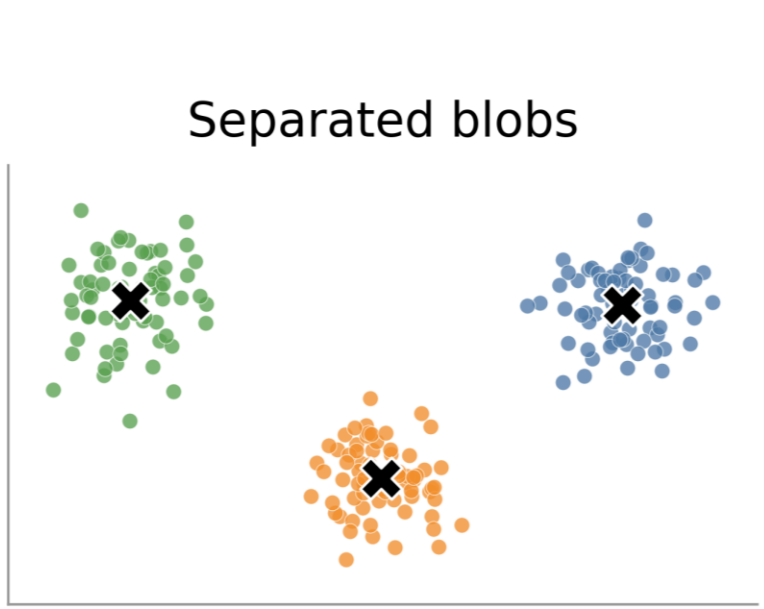


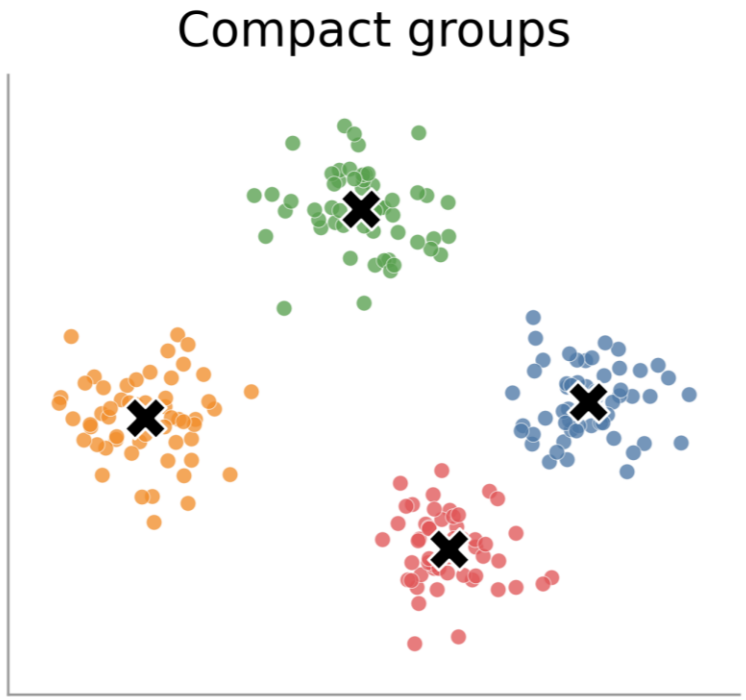


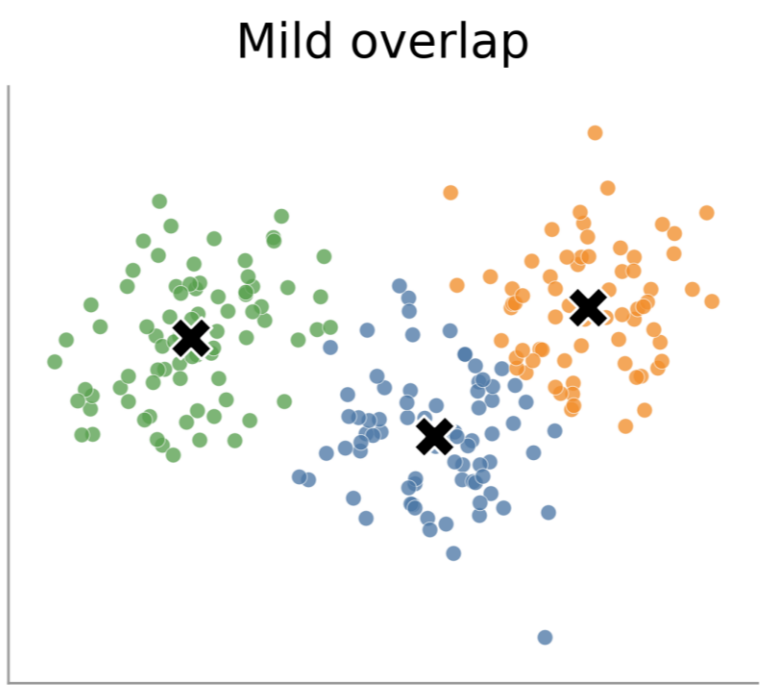




Where k-means clustering works well

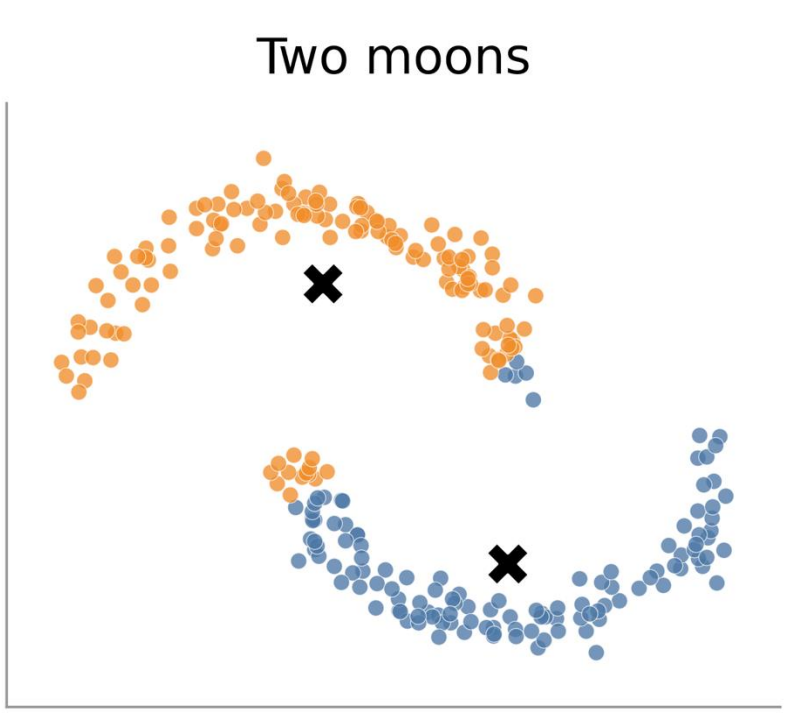


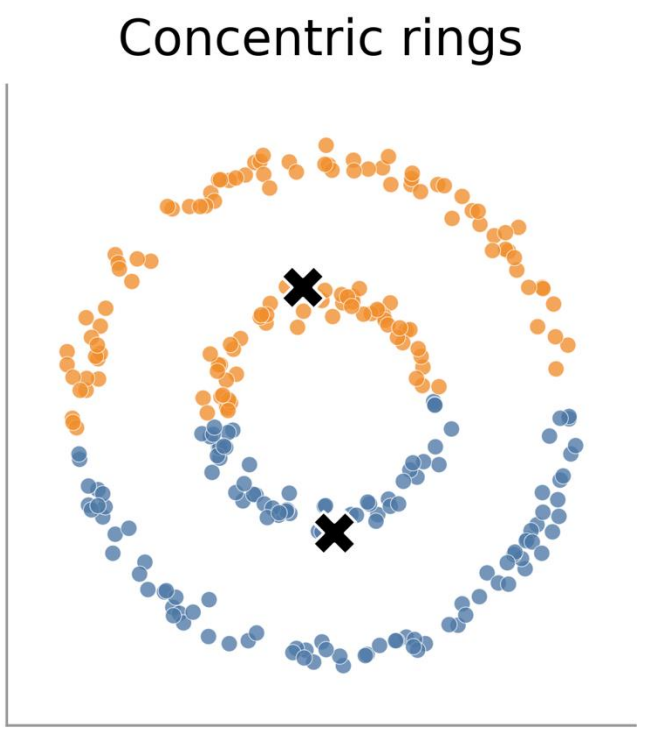


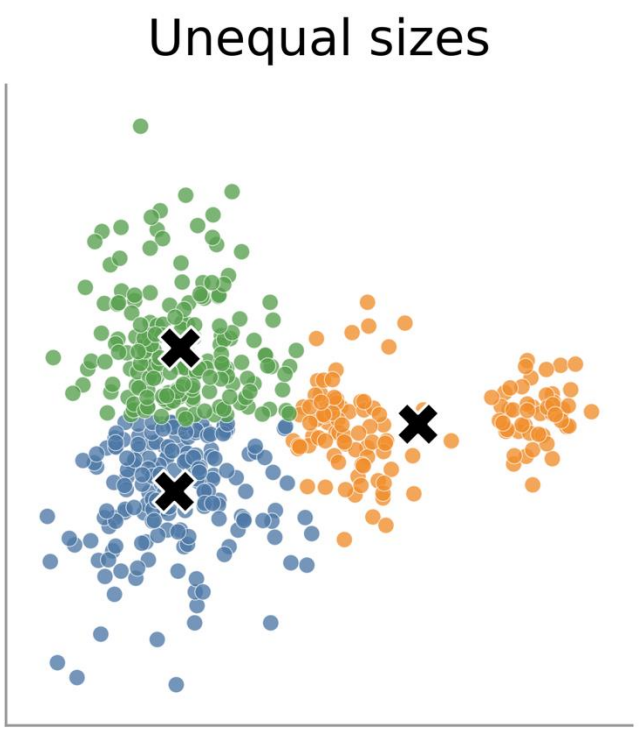


K-means clustering works best when **each cluster is (roughly) spherical**, and the clusters are all of **comparable size and spread**, so that no single cluster dominates the rest

Where k-means clustering *struggles*







It fails in the exact opposite cases: **oddly shaped, uneven clusters**

...Let's introduce some additional methods to deal with this...

Questions?

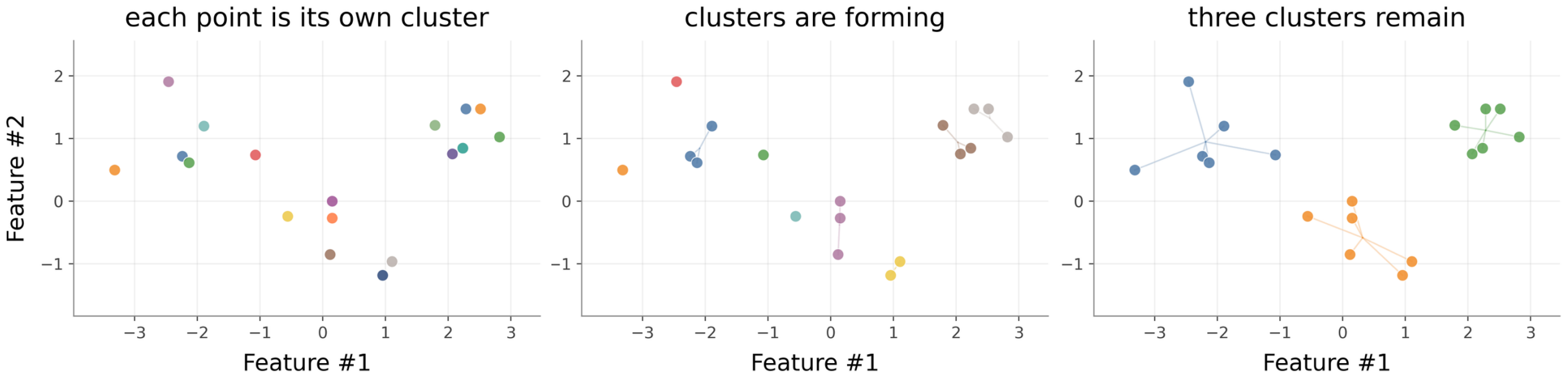
Hierarchical Clustering

Hierarchical clustering: Bottom-up

K-means begins with centroids and iteratively assigns points to build clusters around them.

vs.

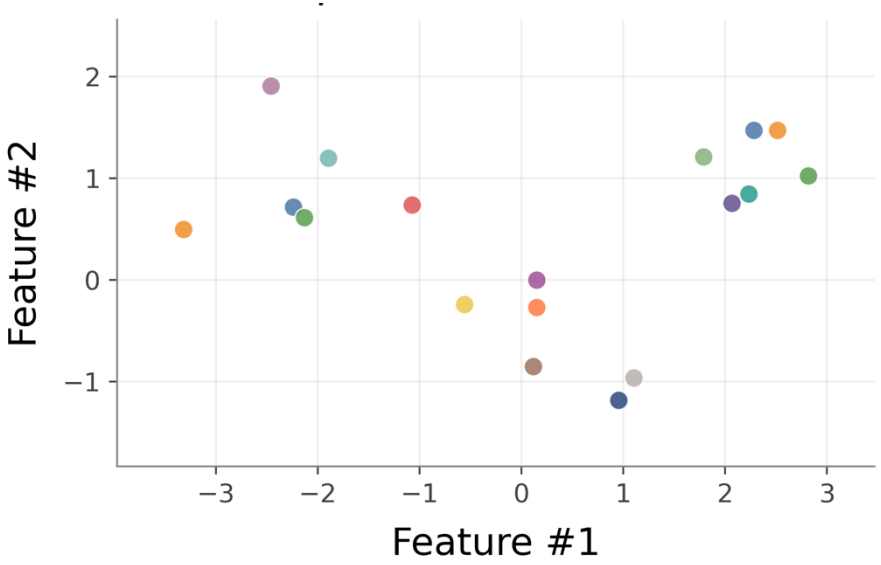
Hierarchical clustering takes the opposite approach, starting with each point as its own cluster and repeatedly merging the closest clusters.



Hierarchical clustering: Step-by-step

1. Start with each point as its own cluster: $\{x_1\}, \{x_2\}, \ldots, \{x_n\}$

2. Compute distances between all clusters: $D(c_i, c_j)$



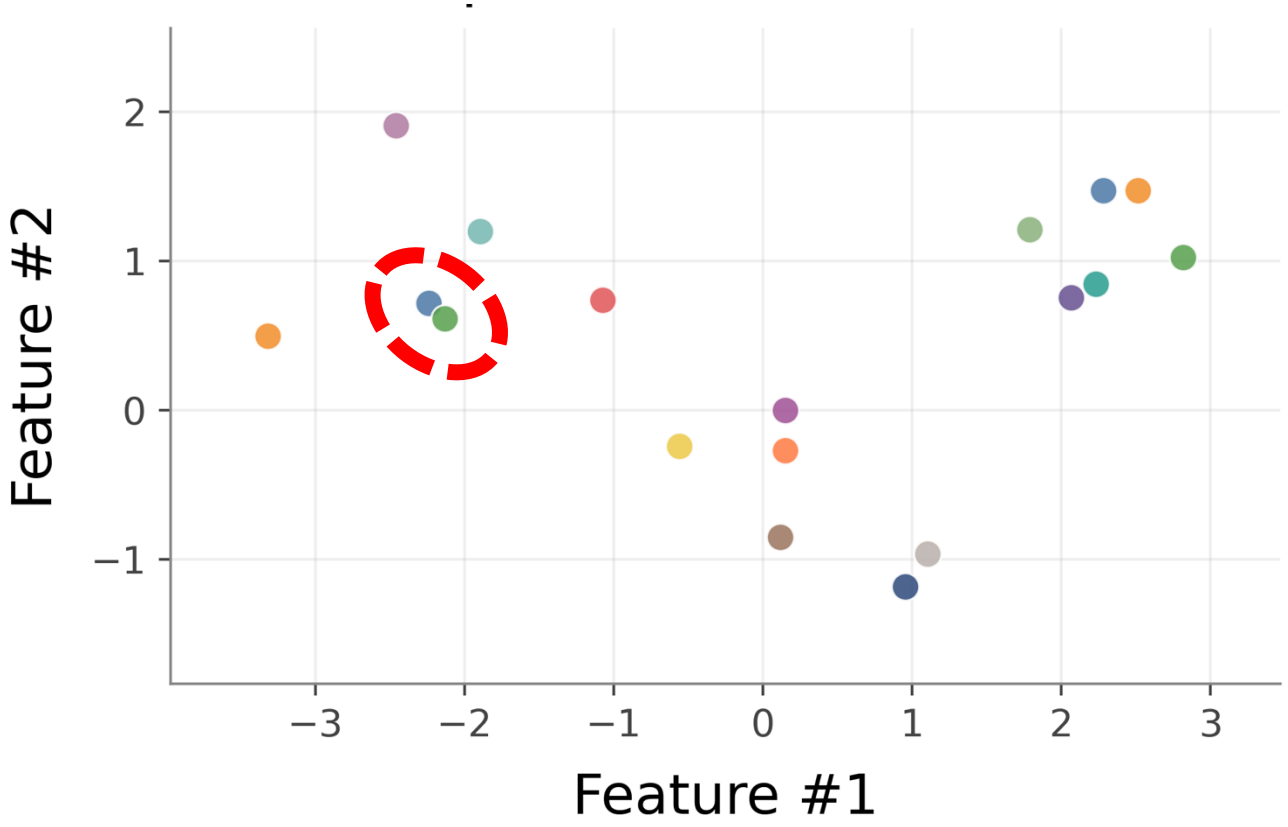
In the first iteration:

- If we have n points, then we have n separate clusters
- Each cluster is just one point, so the distance is:

$$D(c_i, c_j) = \|x_i - x_j\|^2$$ **(Euclidean distance)**

Hierarchical clustering: Step-by-step

3. Find the two closest clusters and merge them



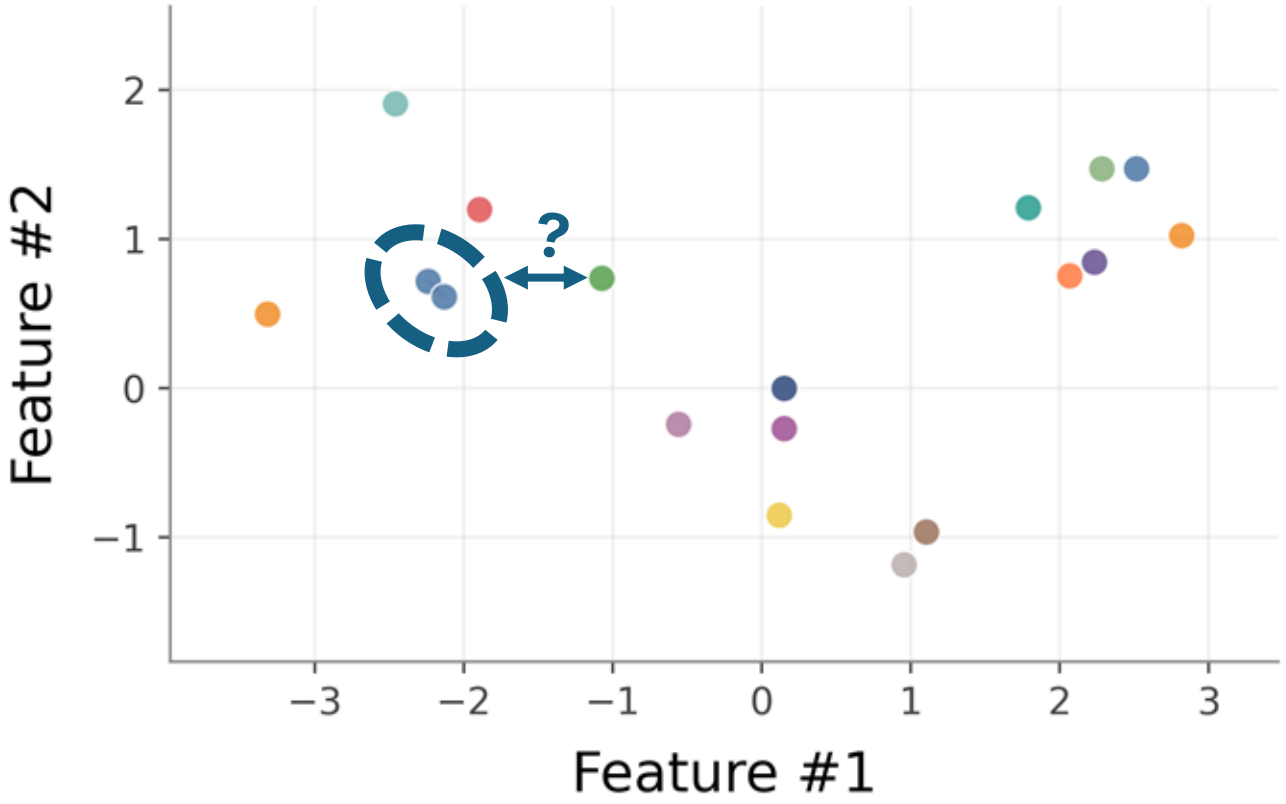
Which two points have the smallest Euclidean distance between them?

→ Merge these into their own cluster

...Then repeat!

Hierarchical clustering: Step-by-step

2. Compute distances between all clusters: $D(c_i, c_j)$

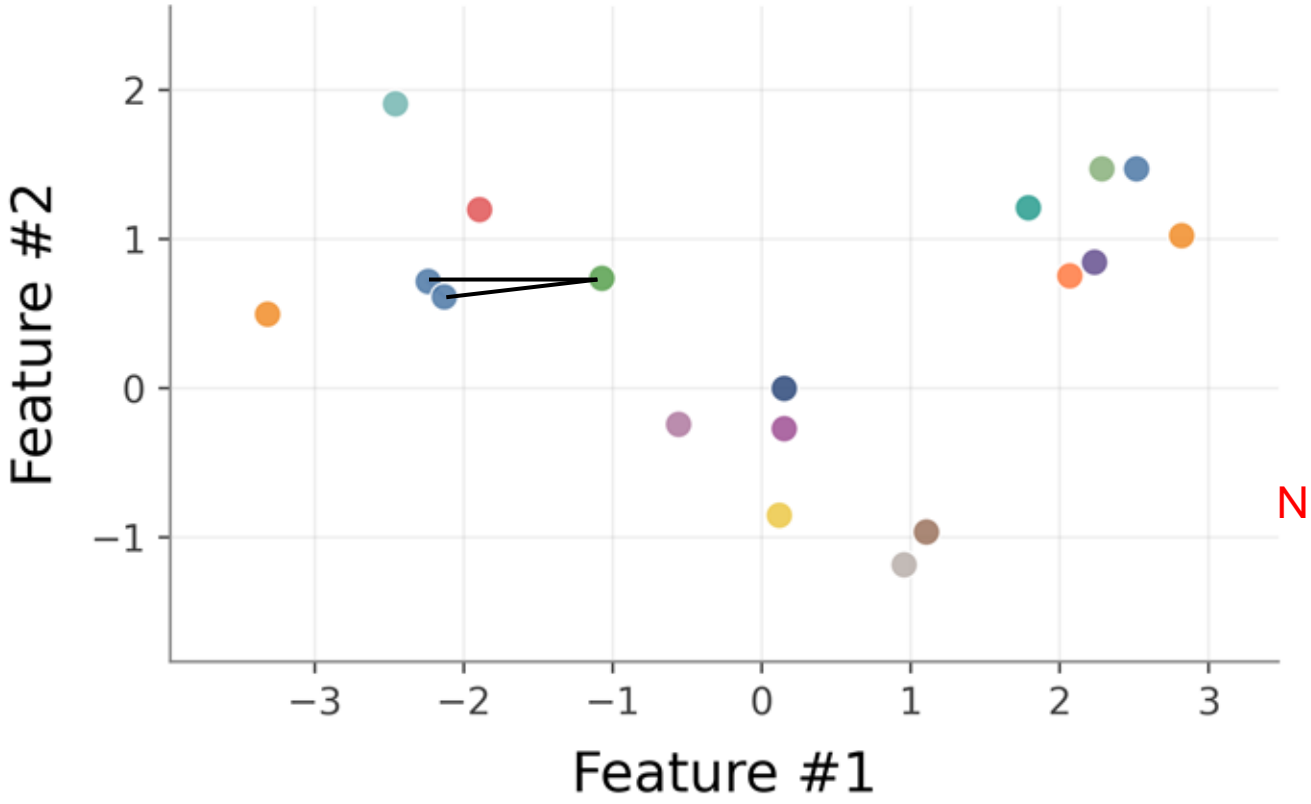


How do we compute the distance between clusters that have more than one point in them? **Options:**

1. Look for the ***smallest*** distance between points
2. Look for the ***largest*** distance between points
3. ***Average*** over all pairwise distances

Hierarchical clustering: Step-by-step

2. Compute distances between all clusters: $D(c_i, c_j)$



Do this for all clusters and again identify the the smallest distance (may or may not involve the one we already made)

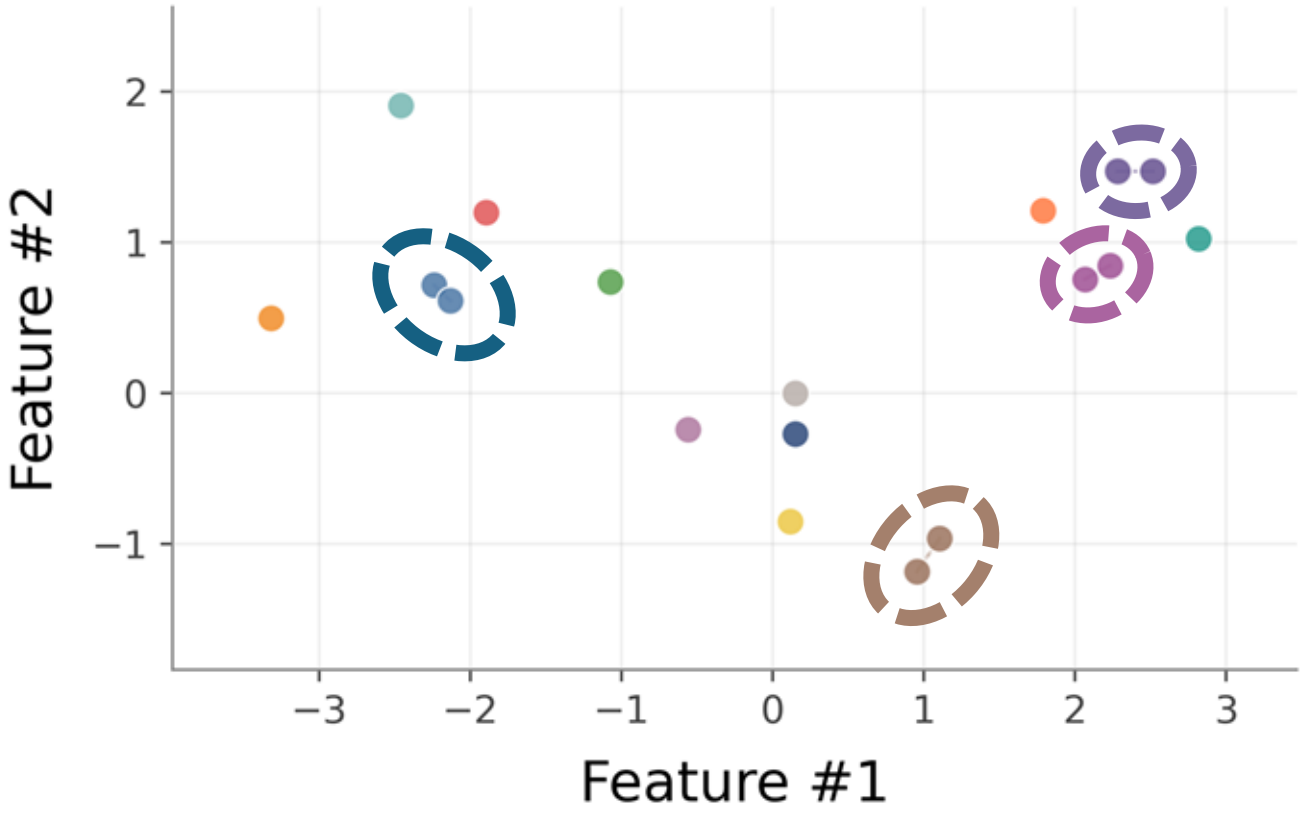
$$D(A,B) = \frac{1}{|A||B|} \sum_{i \in A} \sum_{j \in B} \|x_i - x_j\|$$

Number of pts in each cluster

3. *Average* over all pairwise distances

Hierarchical clustering: Step-by-step

3. Find the two closest clusters and merge them



Compute distances between clusters

Merge nearby clusters

Eventually, smaller clusters will begin to merge together into larger ones…

Hierarchical clustering: Step-by-step

1. Start with each point as its own cluster: $\{x_1\}, \{x_2\}, \ldots, \{x_n\}$

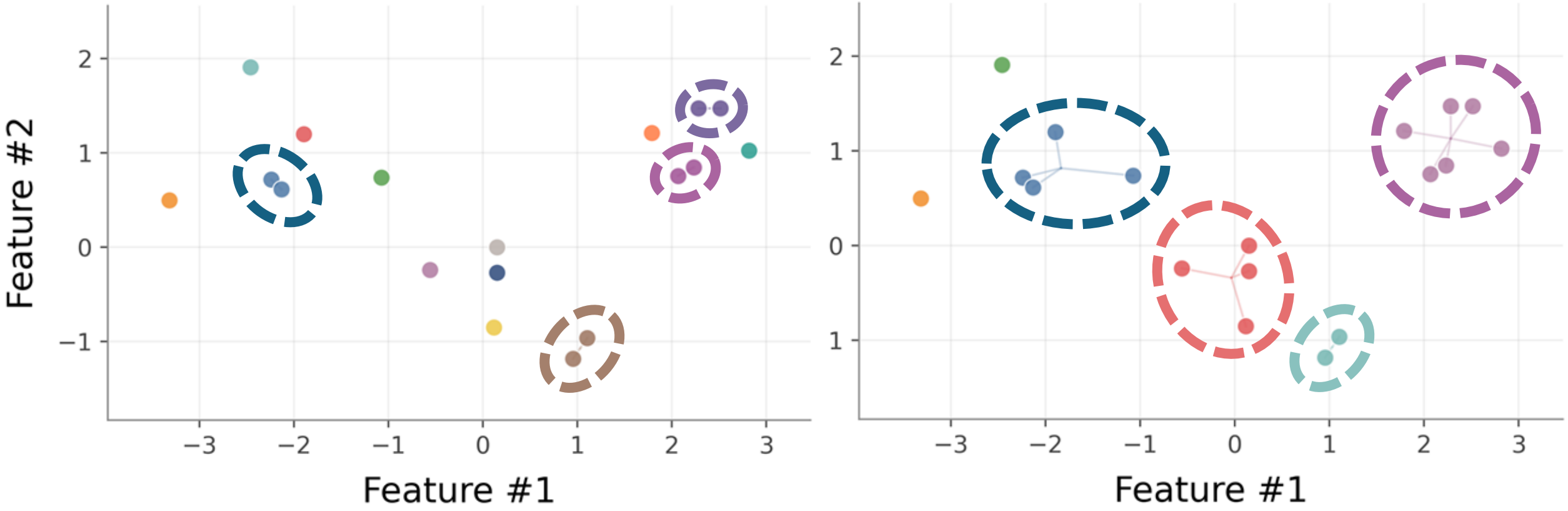
2. Compute distances between all clusters: $D(c_i, c_j)$

3. Find the two closest clusters and merge them

Repeat until we have just one cluster
(we'll separate into more clusters later)

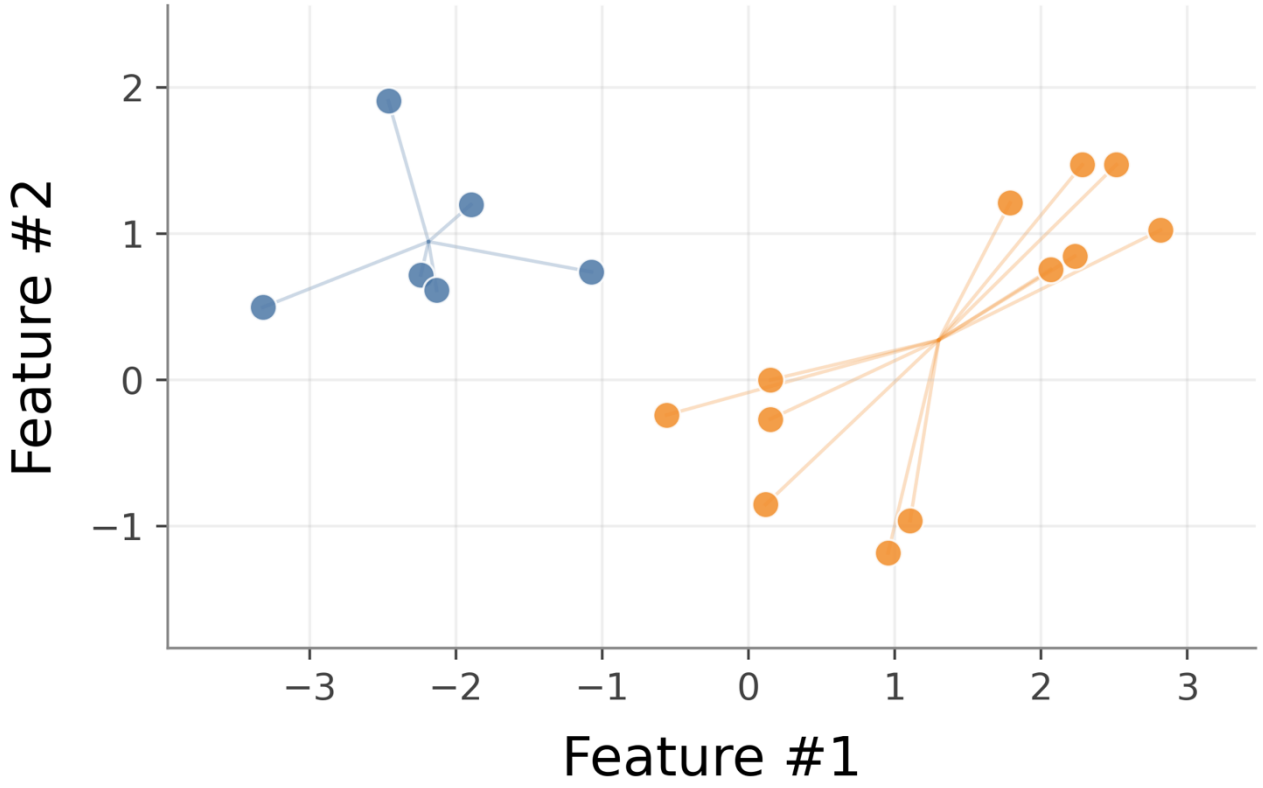
Hierarchical clustering: Step-by-step

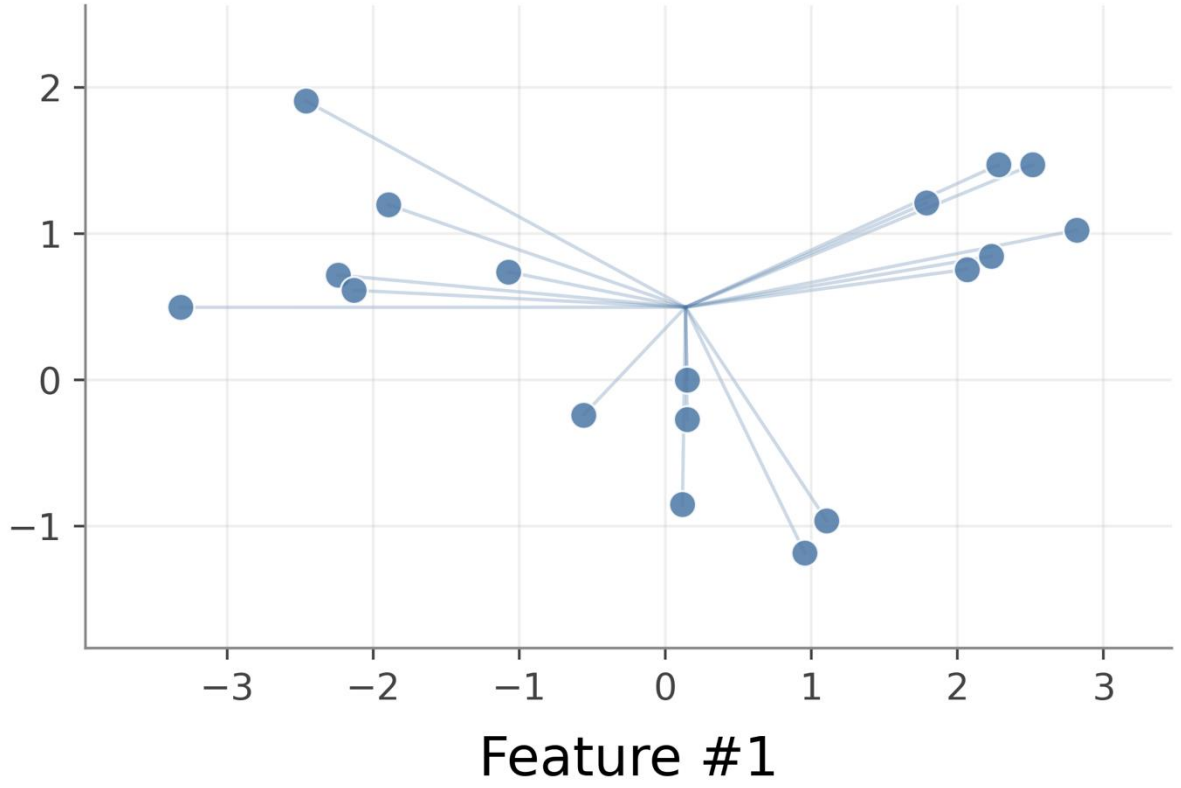
As we continue this process, we get **fewer and larger clusters**



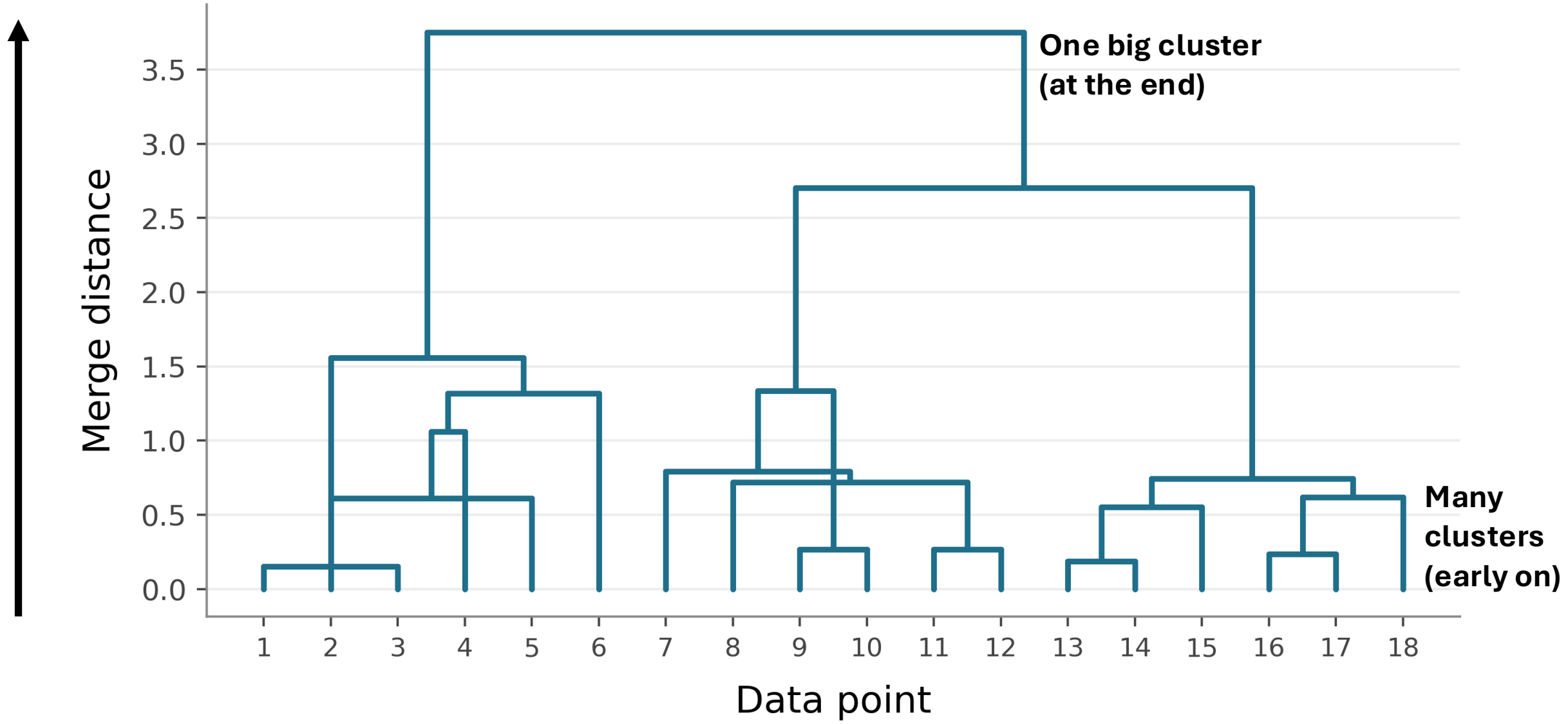
Hierarchical clustering: Step-by-step

Eventually **all points are merged**, and we are left with **one cluster**

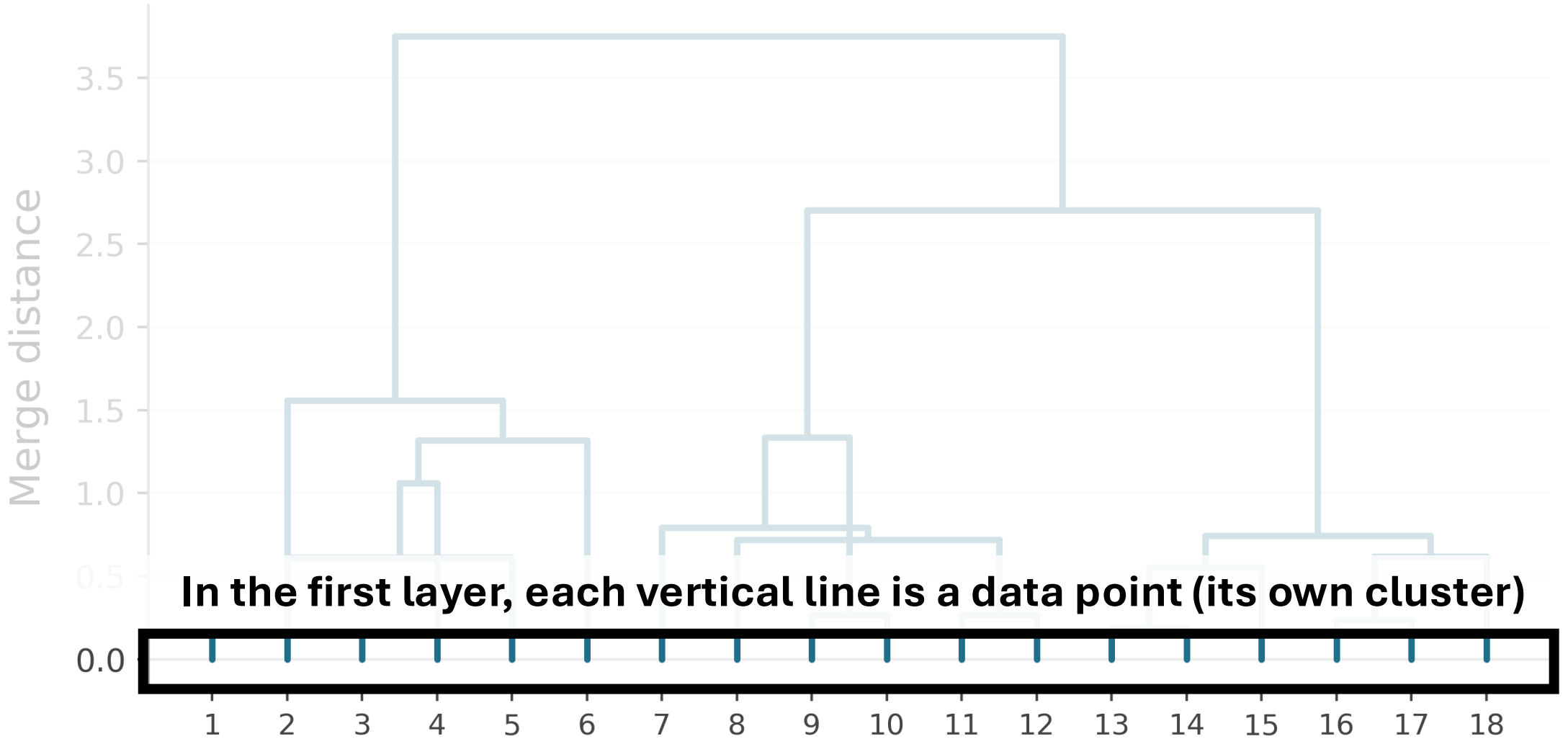




Hierarchical clustering: The dendrogram



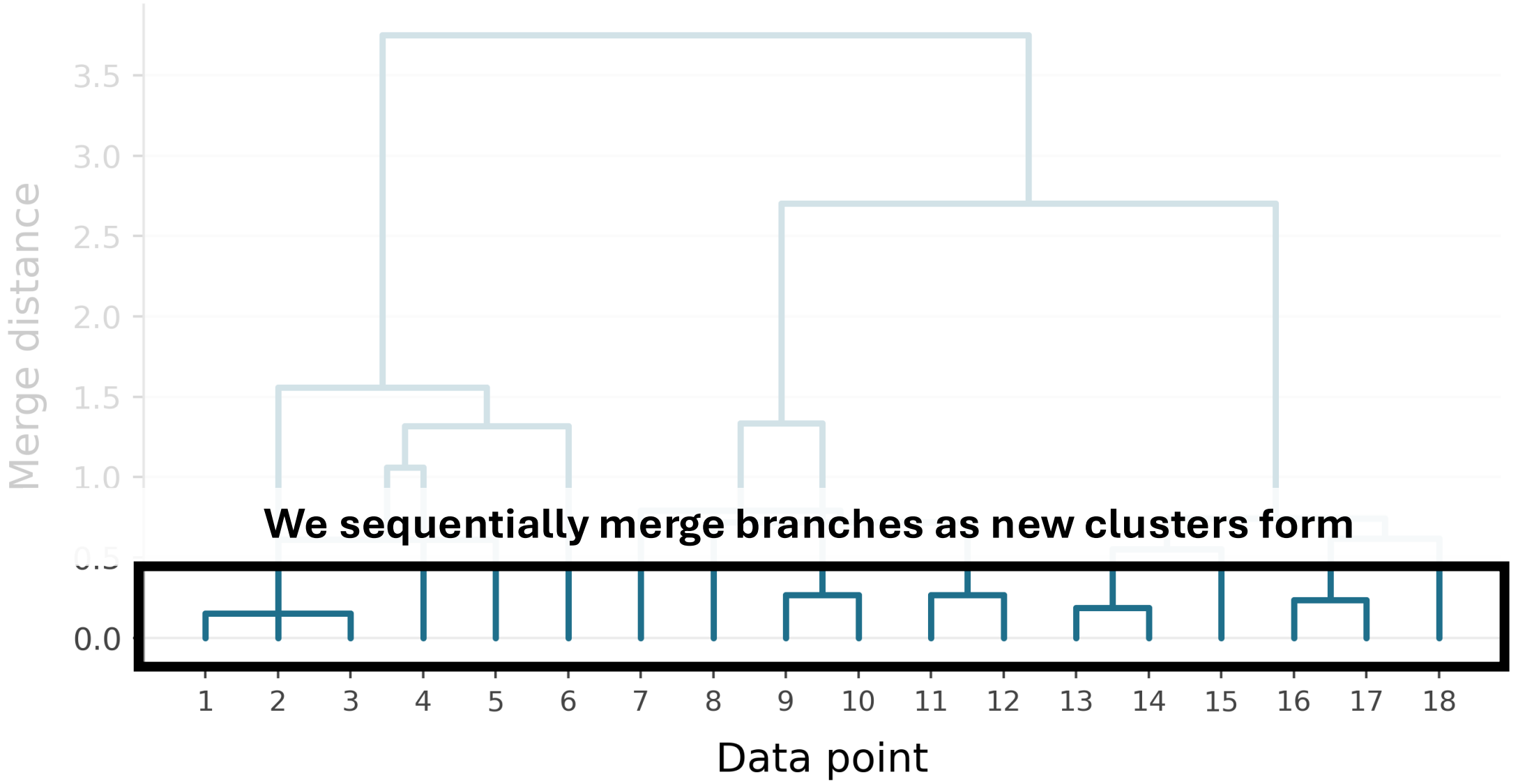
Hierarchical clustering: The dendrogram



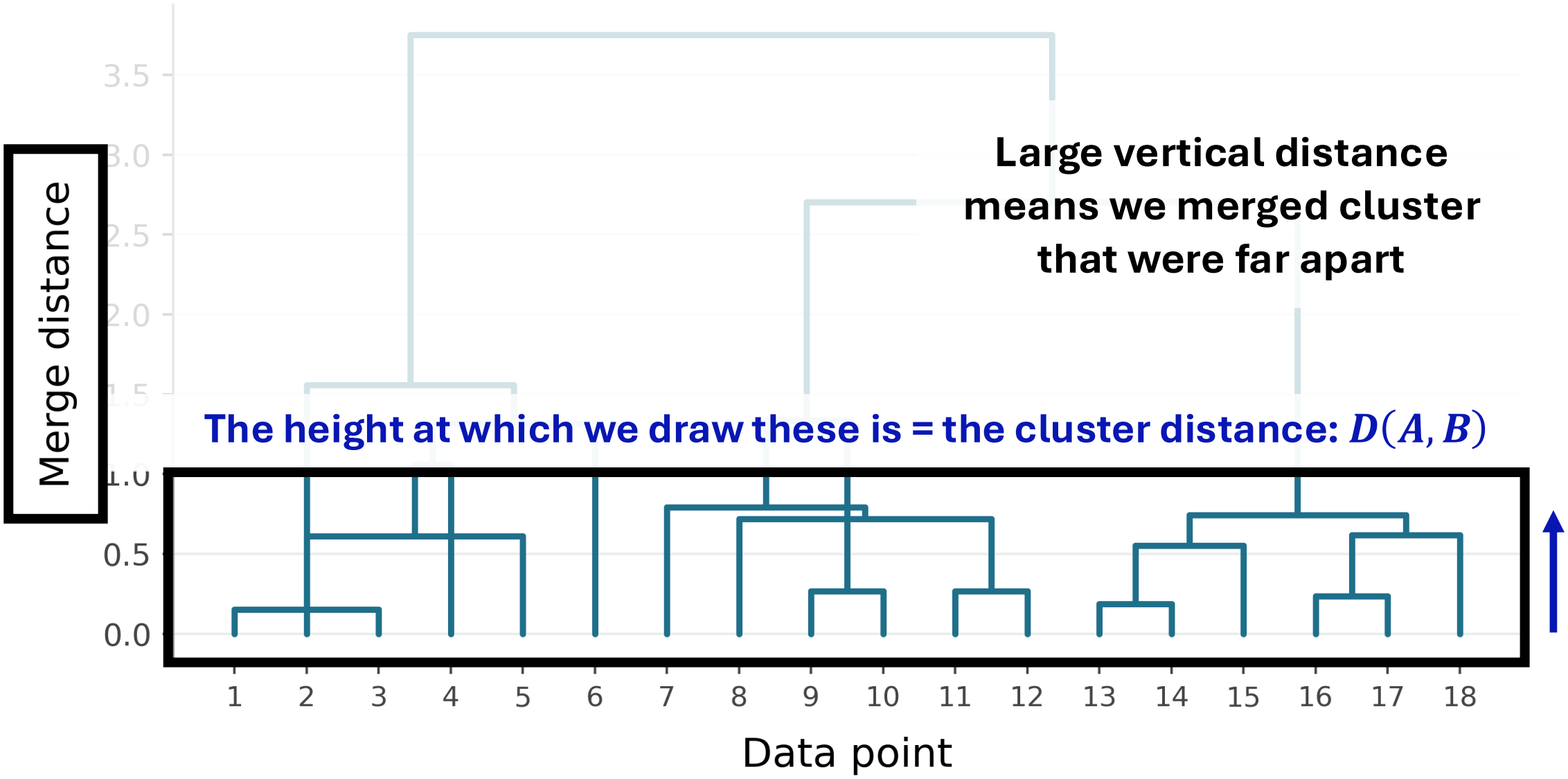
In the first layer, each vertical line is a data point (its own cluster)

***x*-value has no meaning**

Hierarchical clustering: The dendrogram



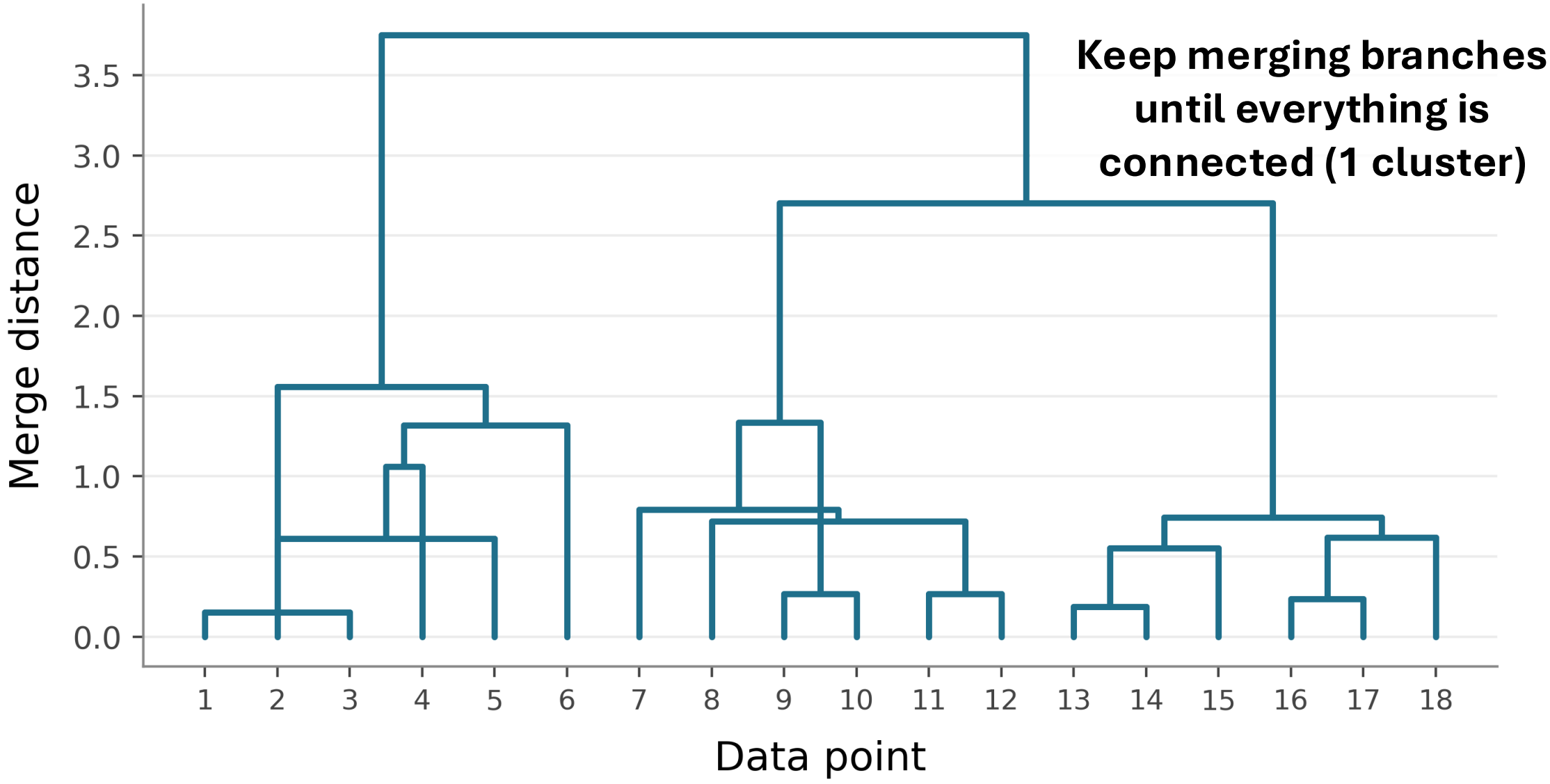
Hierarchical clustering: The dendrogram



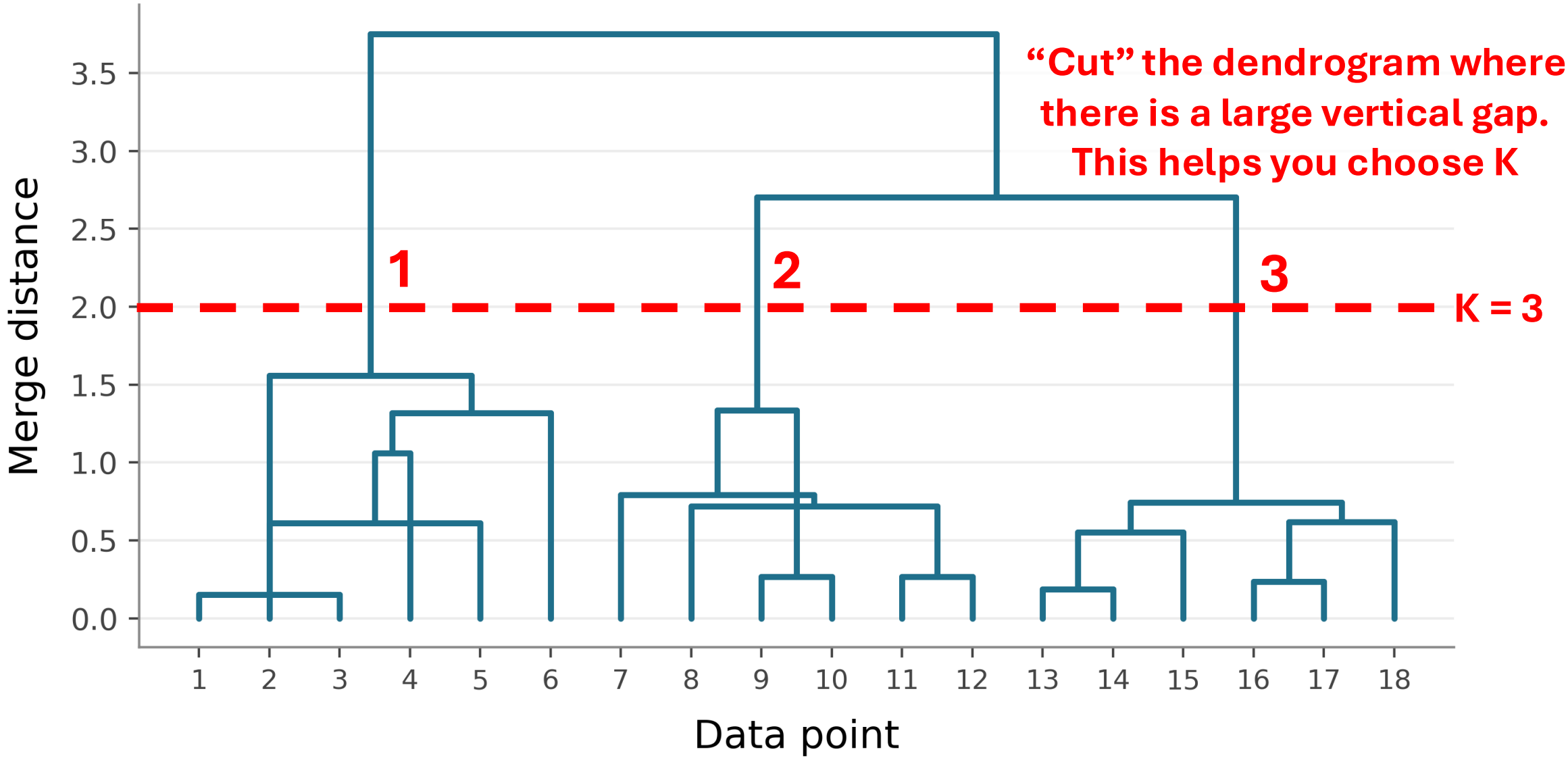
Large vertical distance means we merged cluster that were far apart

The height at which we draw these is = the cluster distance: $D(A, B)$

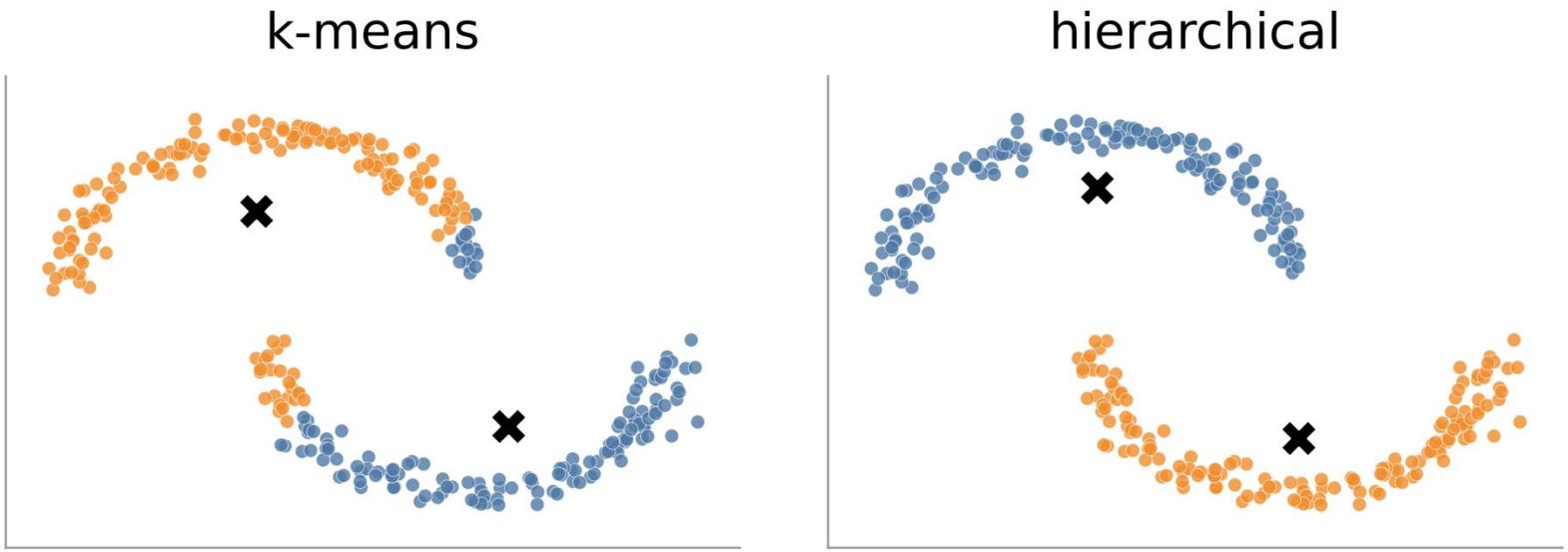
Hierarchical clustering: The dendrogram



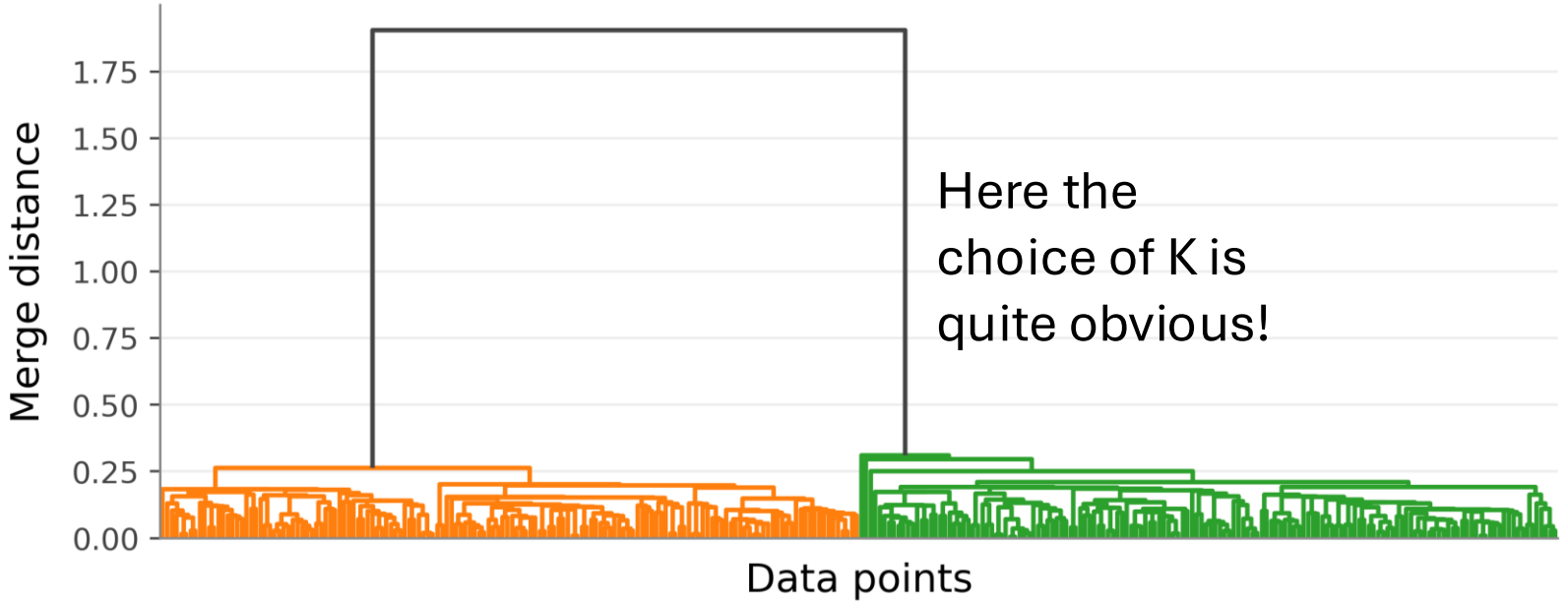
Hierarchical clustering: The dendrogram



Pros of hierarchical clustering



Hierarchical clustering generally handles "curved" data better than K-means clustering

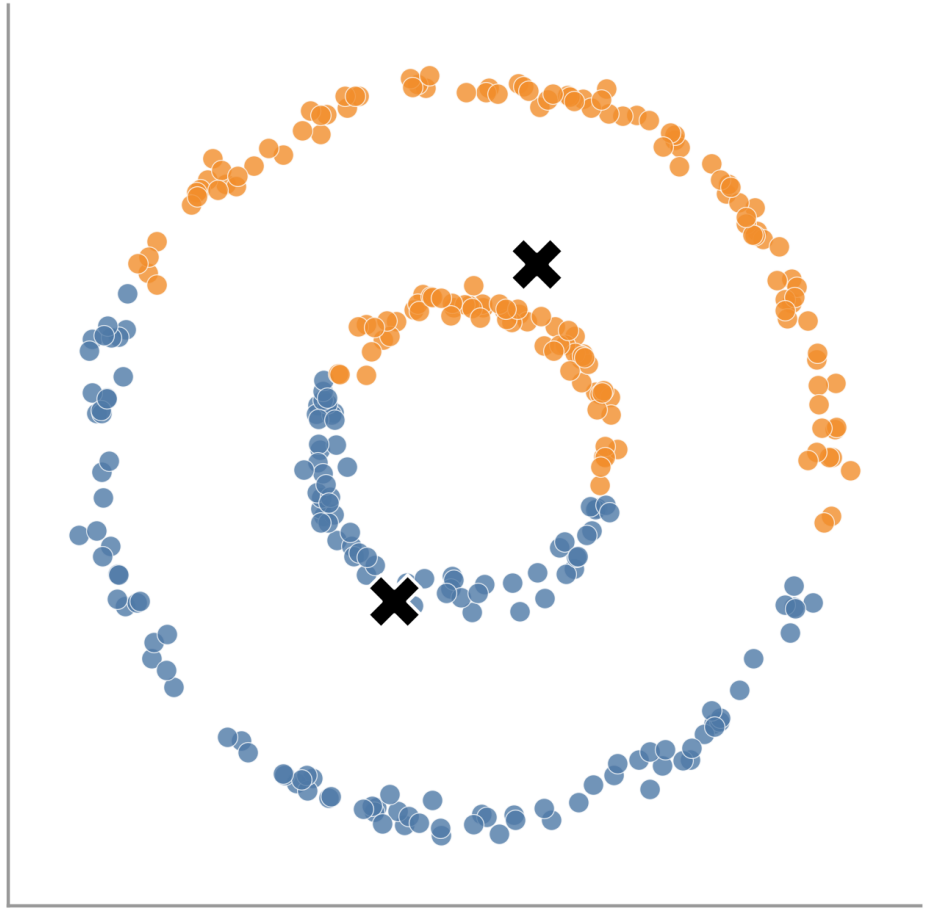


Here the choice of K is quite obvious!

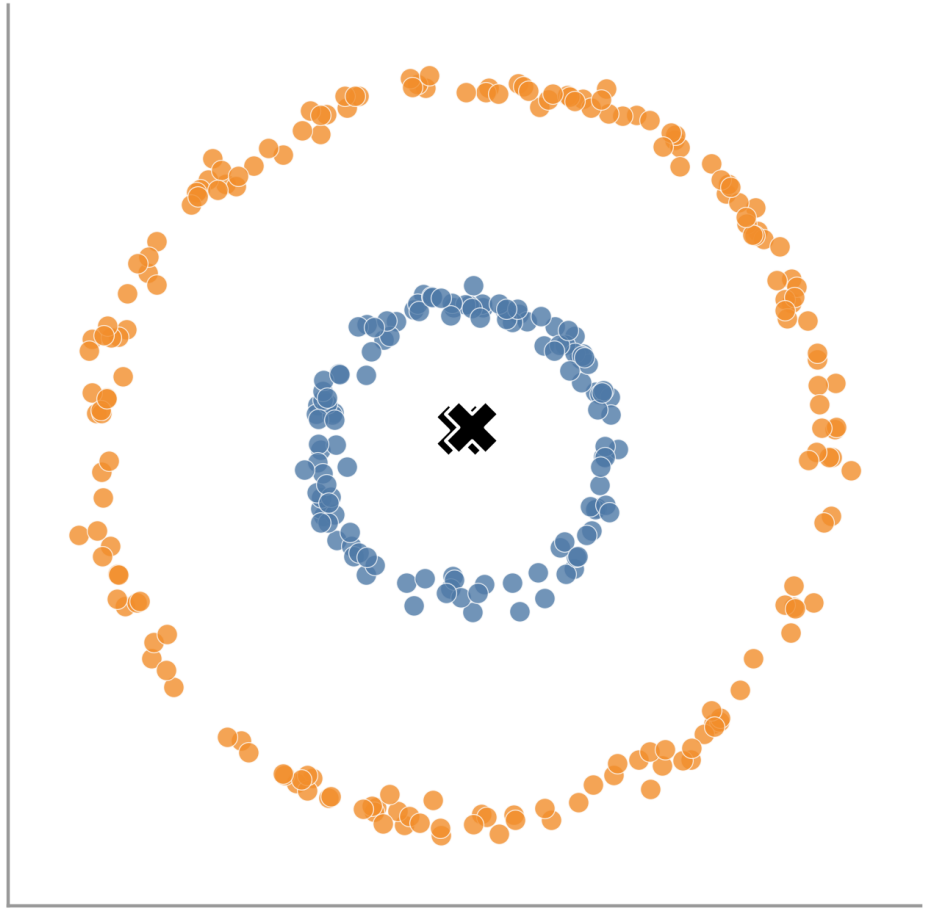
We also don't have to choose K beforehand – **we build all clusters first and decide K after**

Pros of hierarchical clustering

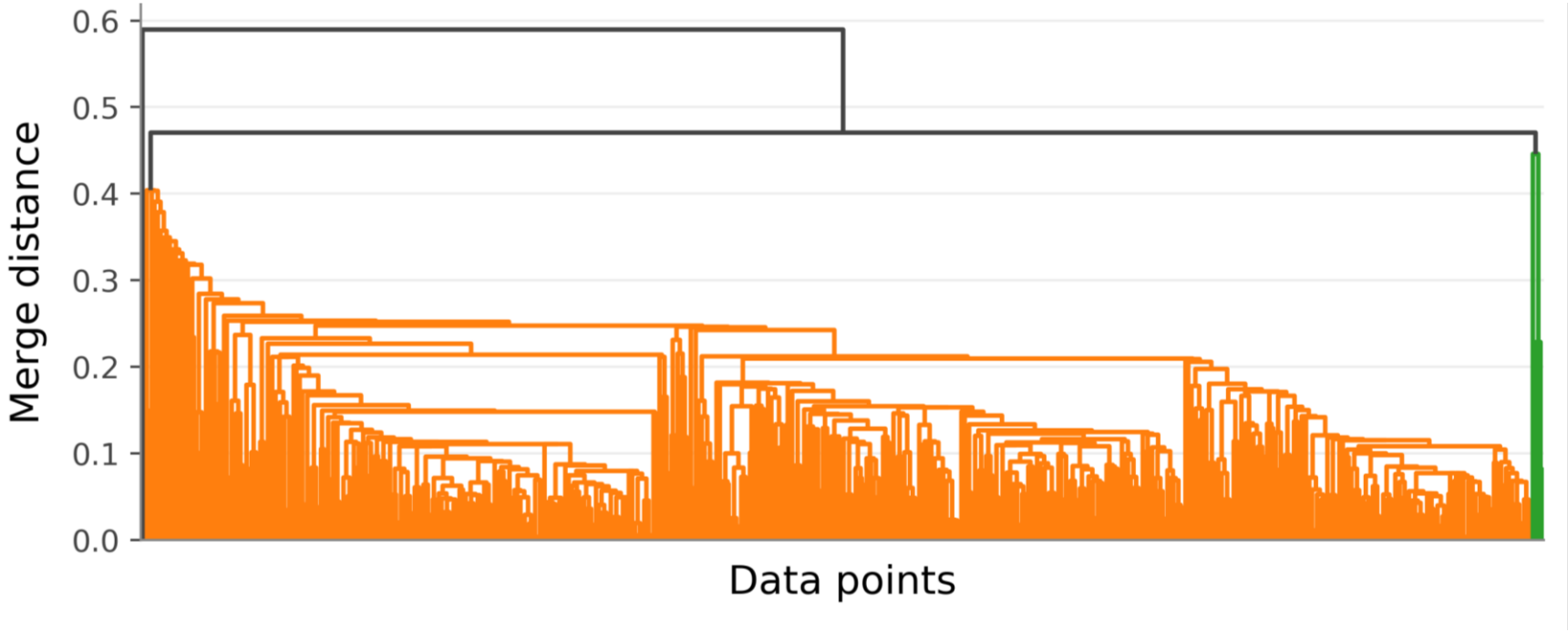
k-means



hierarchical

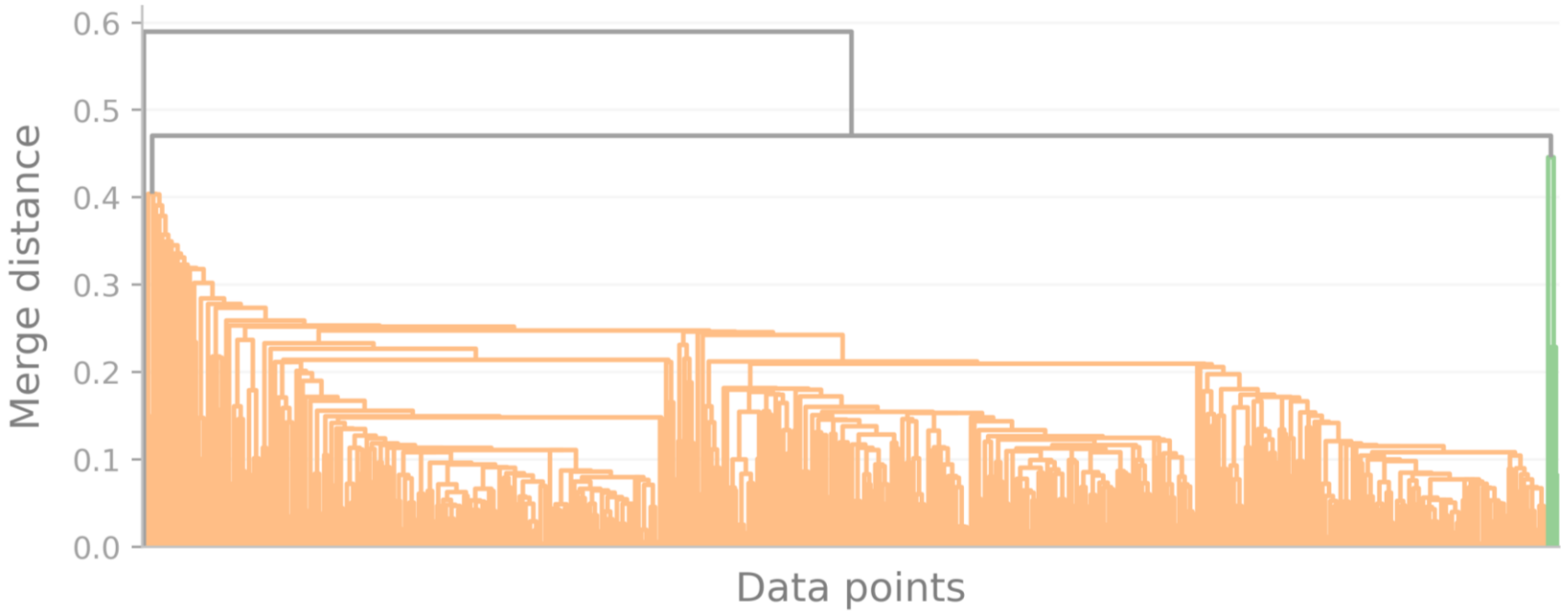


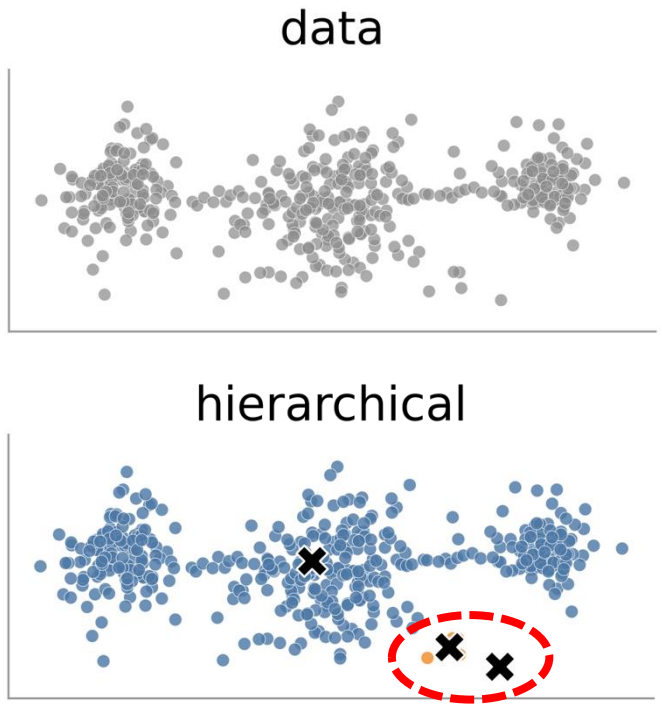
Cons of hierarchical clustering



These dendrograms can get crazy…Because we build the clusters piece-by-piece, hierarchical clustering **scales poorly to large data**: $O(n^2 - n^3)$ vs. $O(n)$ for K-means

Cons of hierarchical clustering





These dendrograms can get crazy…Because we build the clusters piece-by-piece, hierarchical clustering **scales poorly to large data**: $O(n^2 - n^3)$ vs. $O(n)$ for K-means

Hierarchical clustering **performs poorly when cluster densities differ substantially**, which can often be the case when the **data is noisy…**

Questions?

Density-Based Clustering

DBSCAN: clusters as **dense regions**

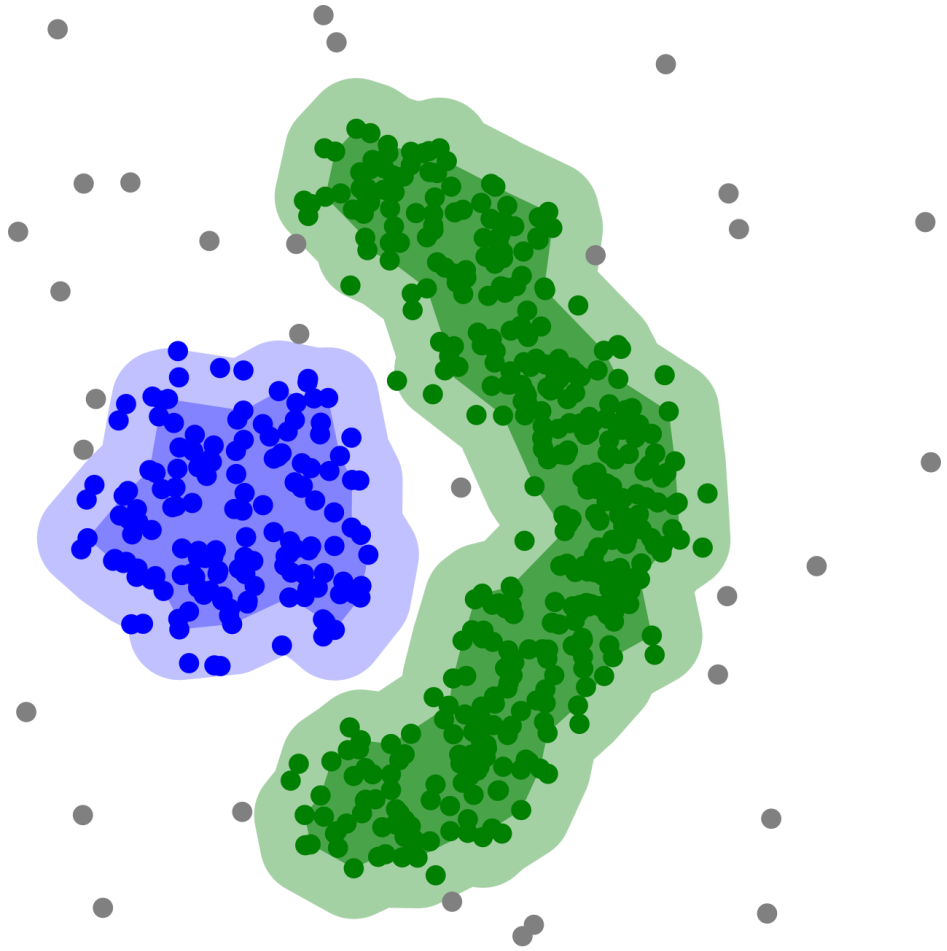
Density-based clustering: DBSCAN

DBSCAN = Density-Based Spatial Clustering of Applications with Noise

Two key ideas:

A cluster is a *dense* region of points

Sparse points in between are *noise*

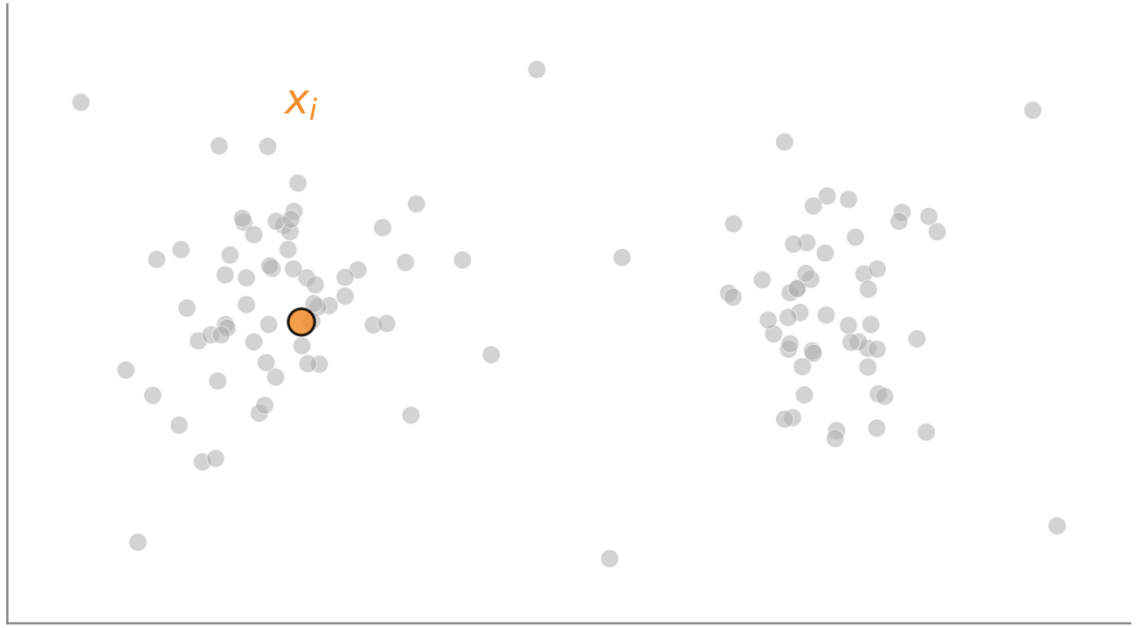


DBSCAN: Step-by-step

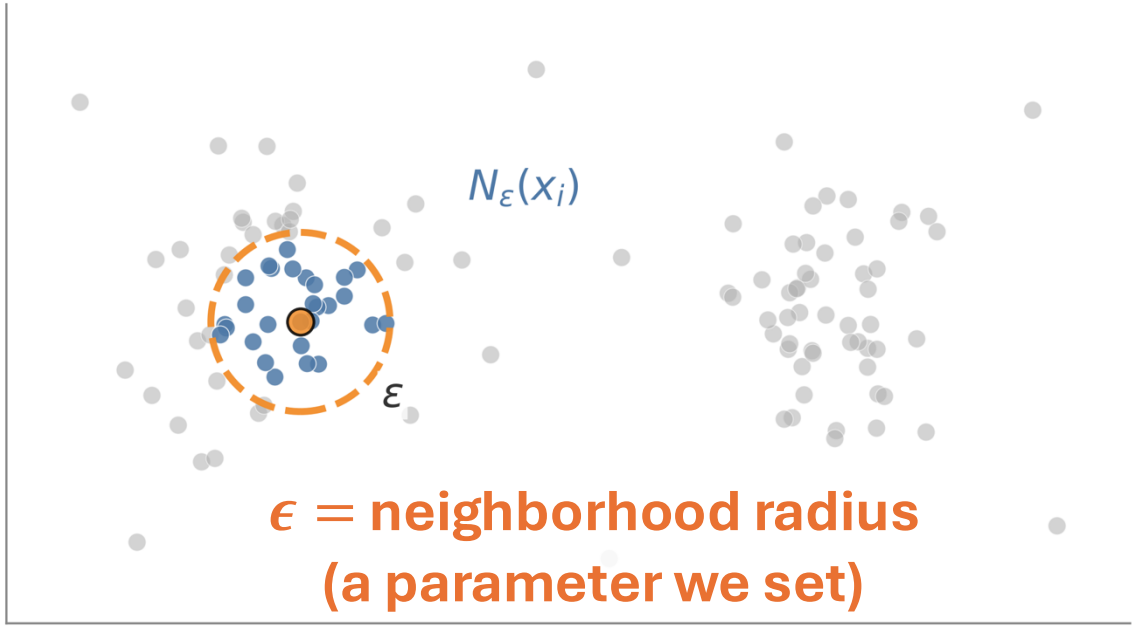
1. Choose a random point: x_i

2. Find all points within the neighborhood (ϵ) of x_i

1. Choose an unvisited point x_i



2. Find the ε-neighborhood



ϵ = neighborhood radius
(a parameter we set)

DBSCAN: Step-by-step

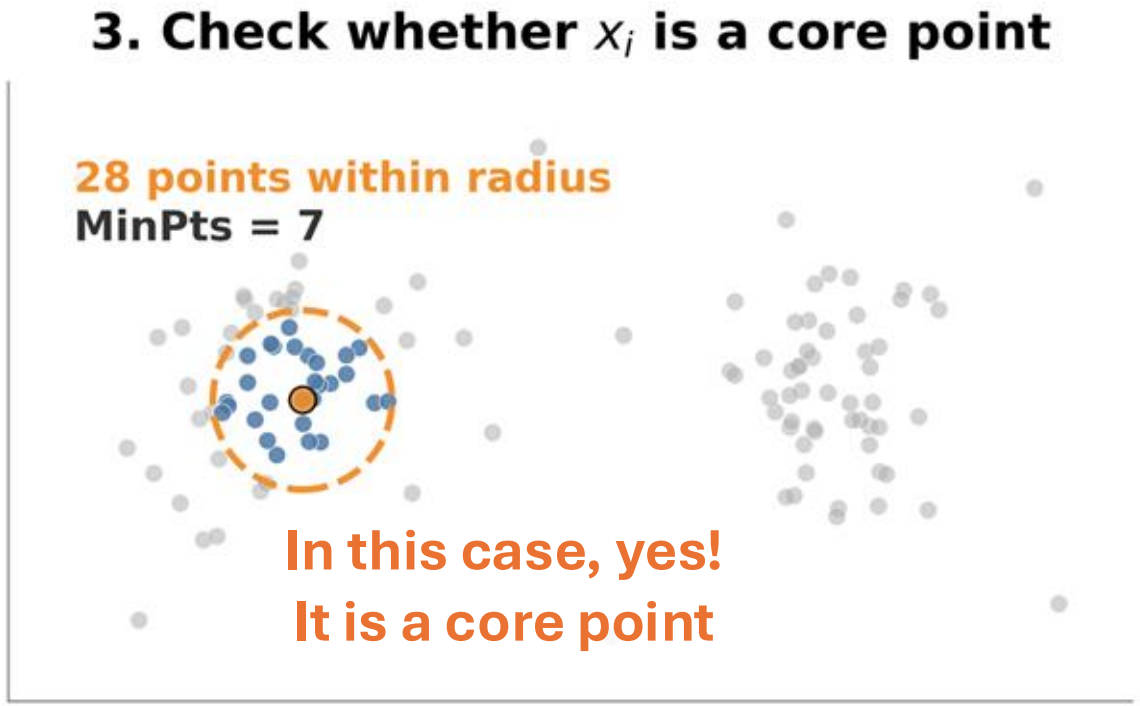
2. Find all points within the neighborhood (ϵ) of x_i

3. Determine whether that same point (x_i) is a "core point" or just noise

In addition to ϵ, we also set **MinPts:** the min number of points needed to form a dense neighborhood

If $N_\epsilon(x_i) \geq \text{MinPts}$, then x_i is considered a **core point**

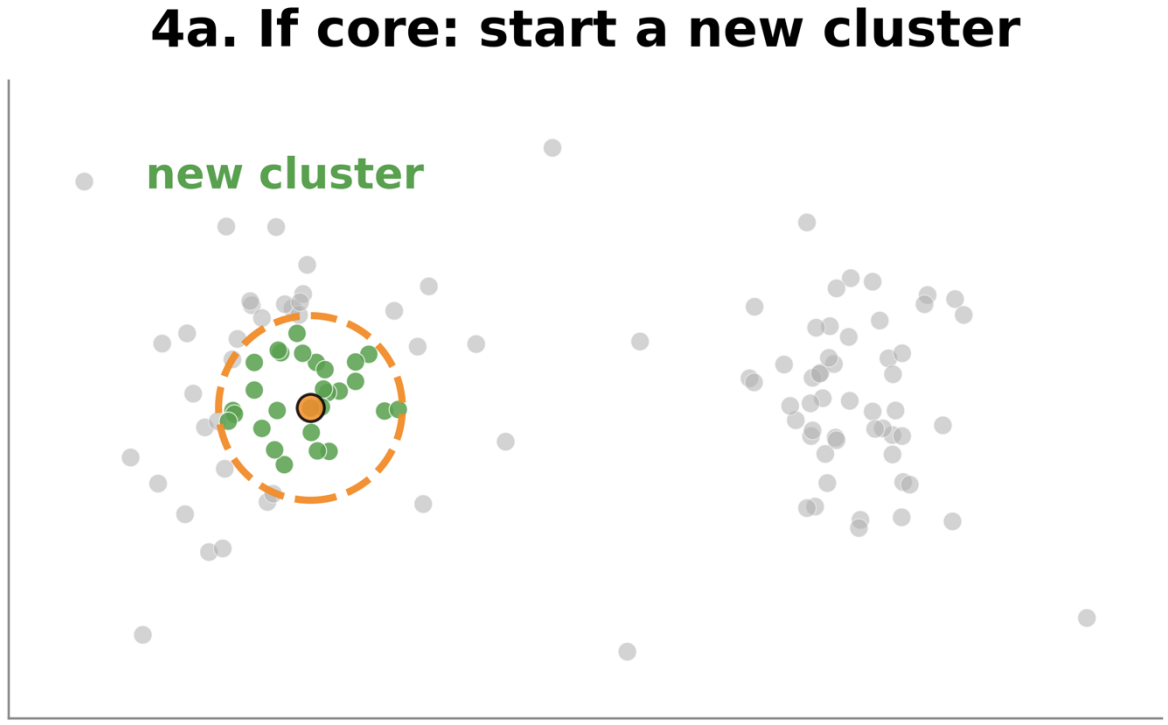
Basically: Is it part of a dense cluster?
If lots of points nearby, yes!

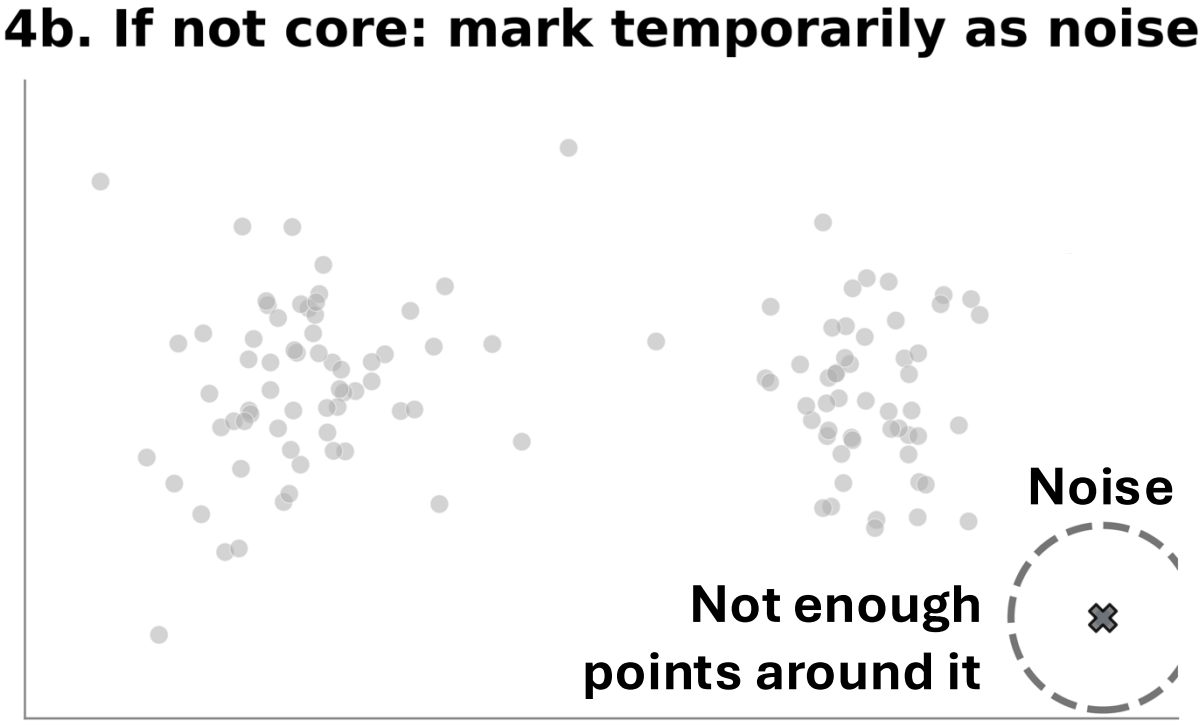


DBSCAN: Step-by-step

4a. If the point *is* a core, then start a new cluster

4b. If the point is *not* core, temporarily mark it as noise





DBSCAN: Step-by-step

5. Expand the cluster. For every neighbor of the core point:

Calculate its own ε-neighborhood. If the new point is also a core point, add its neighbors to the original cluster. Continue until no new reachable points can be added.

5a. Visit neighbors of the core point



5b. Core neighbor expands cluster



DBSCAN: Step-by-step

1. Choose a new, random unvisited point: x_i

2. Find all points within the neighborhood (ϵ) of x_i

3. Determine whether x_i is a core point

4a. If the point *is* a core, then start a new cluster

4b. If the point is *not* core, temporarily mark it as noise

5. Expand the cluster. For every neighbor of the core point:

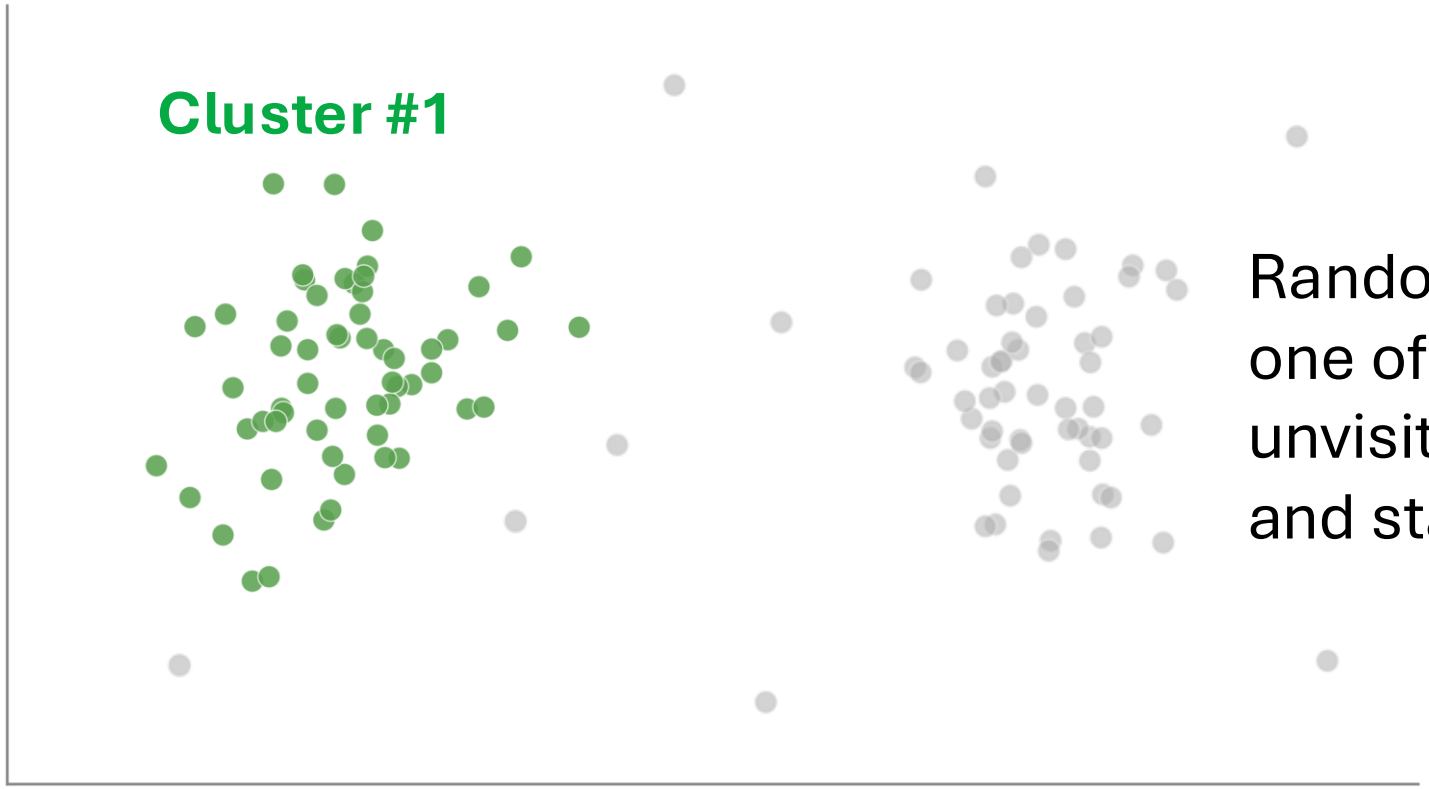
Visit that neighbor. Find its own ε-neighborhood. If that neighbor is also a core point, add its neighbors too. Continue expanding until no new reachable points can be added.

Repeat until every point has been visited

DBSCAN: Step-by-step

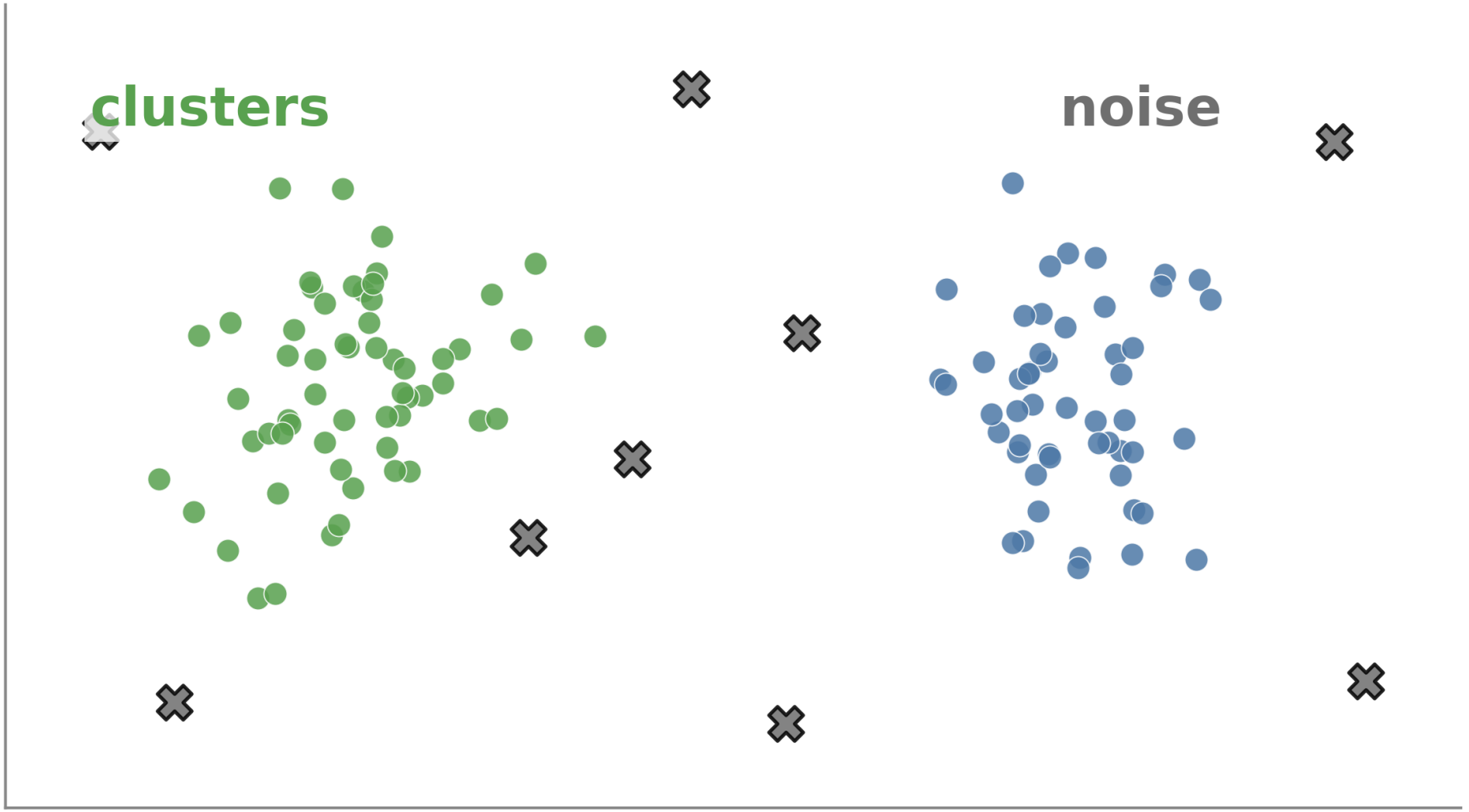
1. Choose a new, random unvisited point: x_i

6. Repeat until all points are visited

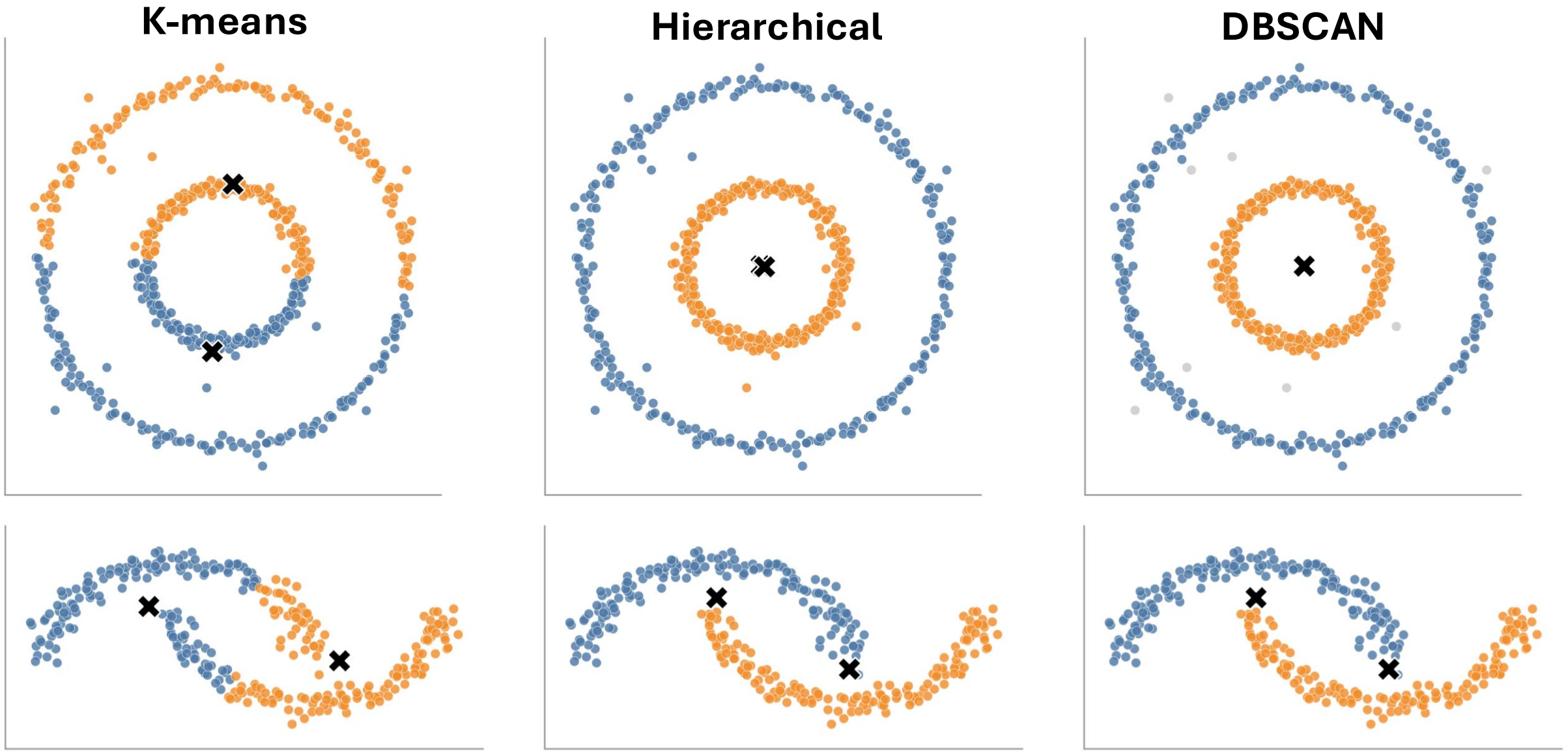


Randomly choose one of these unvisited points and start again!

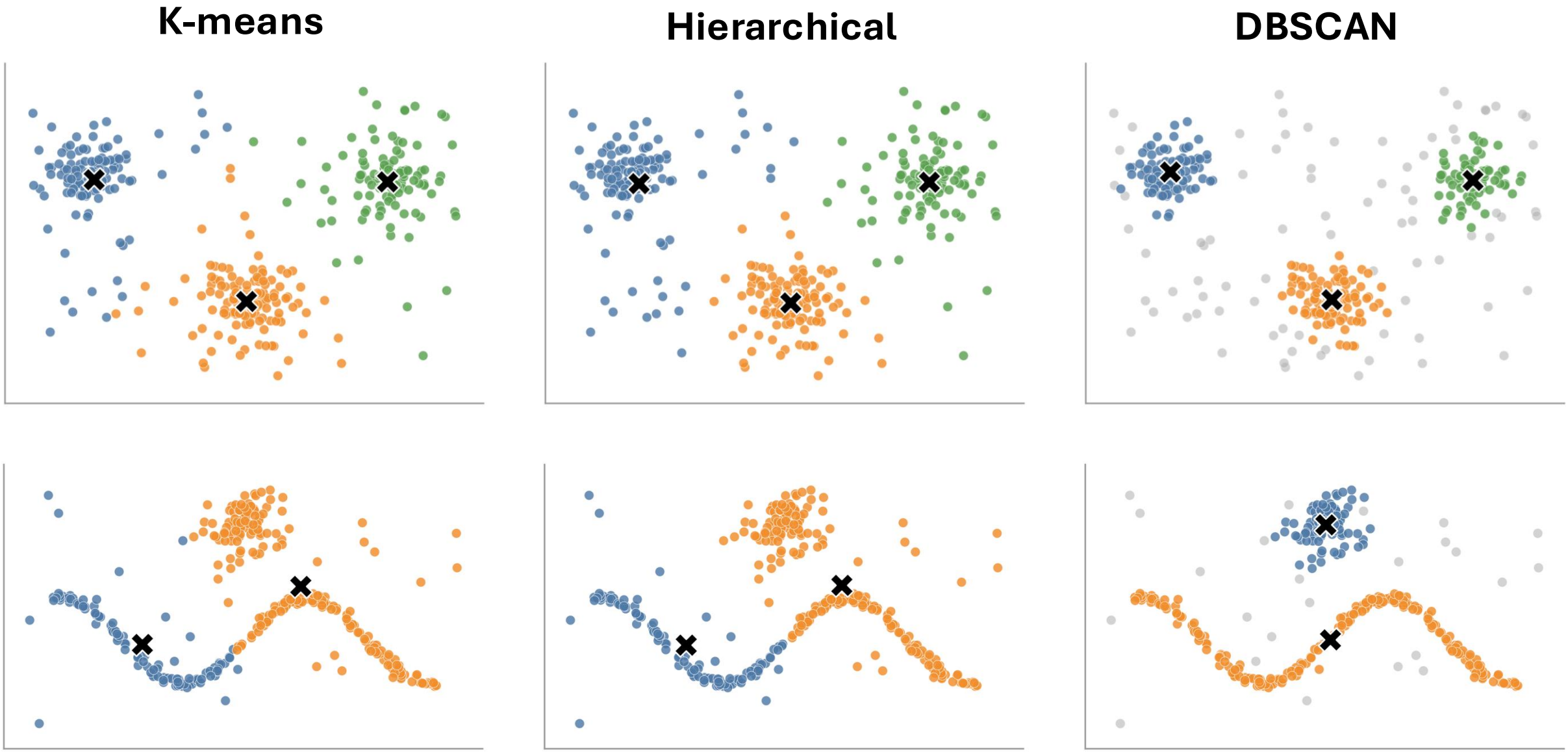
DBSCAN: Final result!



DBSCAN: Some examples



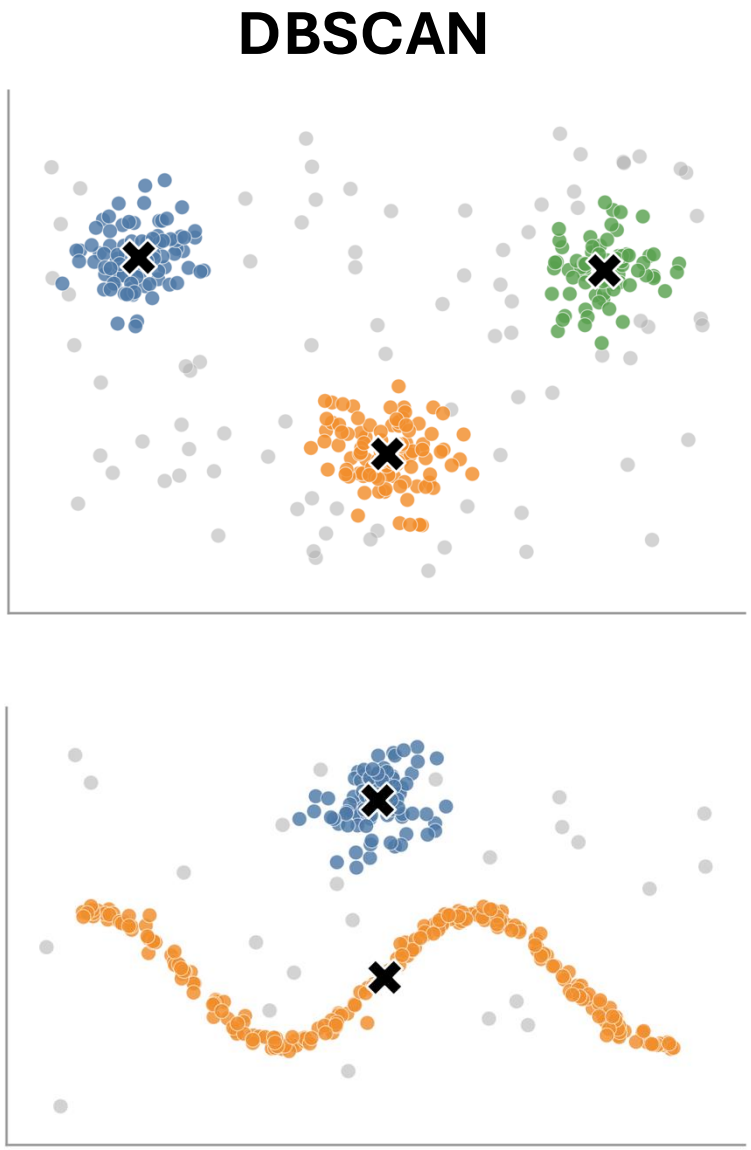
DBSCAN: Some examples



DBSCAN: Pros and Cons

Advantages of DBSCAN:

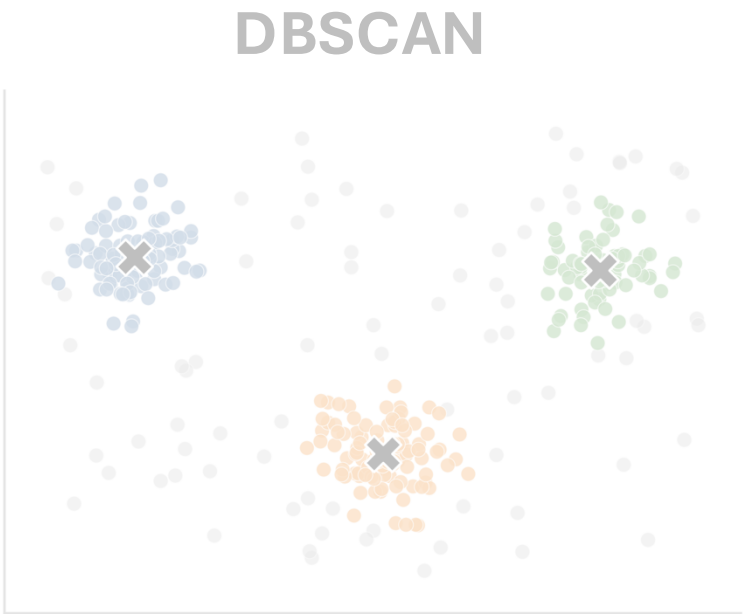
- Performs well on clusters with a variety of shapes as it's only based on density
- Usually performs comparably to hierarchical clustering, but it also **identifies noise – a nice benefit!**
- Unlike k-means and hierarchical clustering, **we do not have to manually choose the number of clusters!**



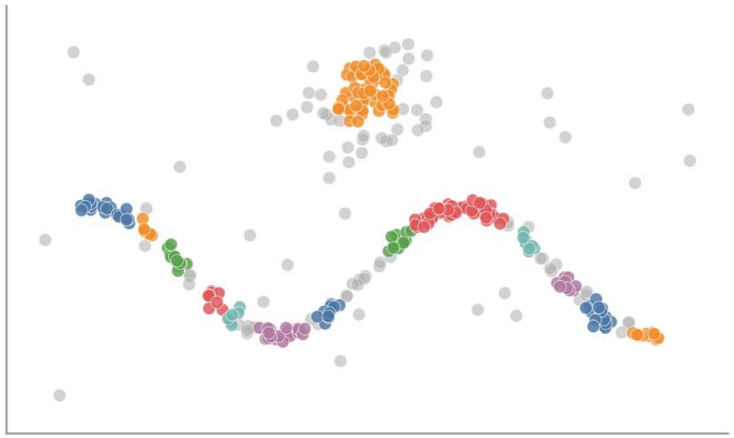
DBSCAN: Pros and Cons

Disadvantages of DBSCAN:

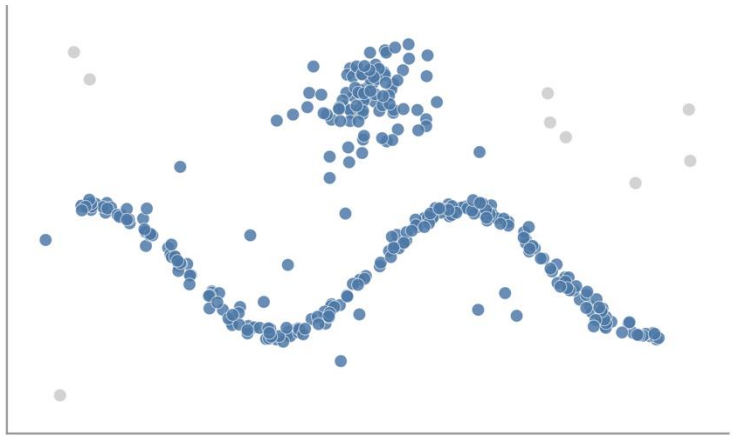
- While we don't have to choose the number of clusters, **we *do* choose ϵ and MinPts…** Choosing the right values can be difficult!
  - Use elbow method to choose radius (ϵ)

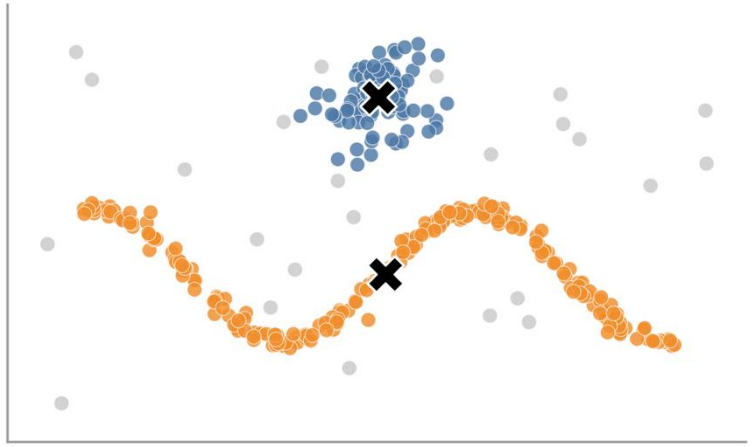


ϵ too small



ϵ too large

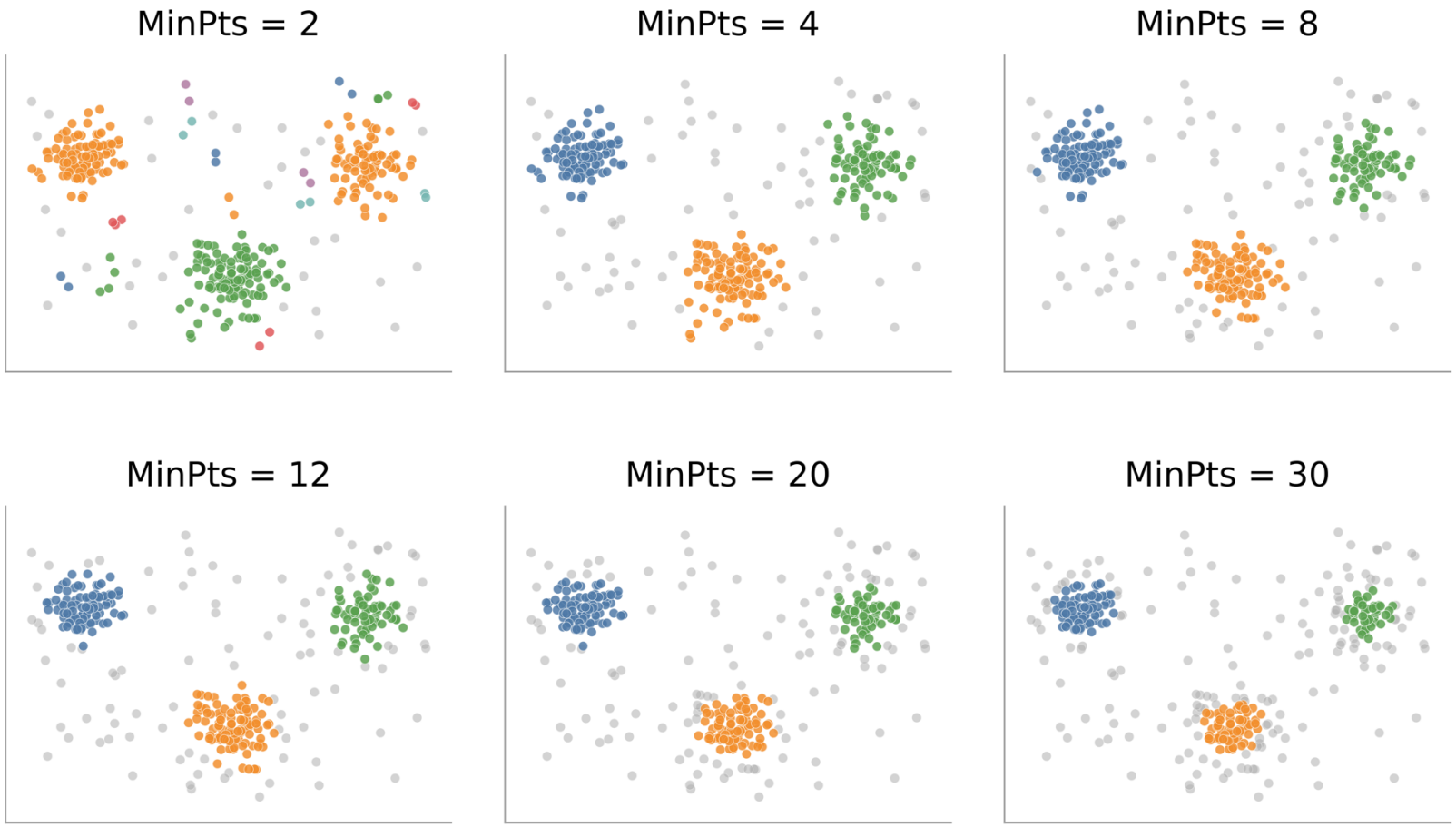




Choosing MinPts

MinPts controls what's considered a cluster *vs.* noise…Up to you!

- A rule of thumb is to set $\mathbf{MinPts} \approx \mathbf{2}\boldsymbol{d}$ where d is the number of dimensions
- So, if we have two features ($d = 2$) then four points is enough to form a cluster
- Usually, never choose $\mathbf{MinPts} < \mathbf{4}$, as this will cause noisy points to become small clusters



Questions?

Which one to use? It depends…

k-means

- Your clusters are spherical and similar in size
- You have an idea of how many clusters there are
- You want a simple and fast algorithm (large data)

Hierarchical

- You are dealing with small data ($\leq$ thousands)
- You don't know how many clusters there are
- You value interpretability (the dendrogram)

DBSCAN

- You have "medium" data (tens of thousands)
- You anticipate oddly shaped clusters
- You want to identify noise/outliers/anomalies

What to know for HW + Exams

- Mostly focus on **k-means clustering**
  - **The meaning of k**
  - **Understanding inertia**
  - **How to choose “the best” k value (silhouette score)**
- Hierarchical and DBSCAN: Just know the concepts