CMPINF 2100
Introduction to Data Centric Computing

Week 08

Introduction to Cluster analysis

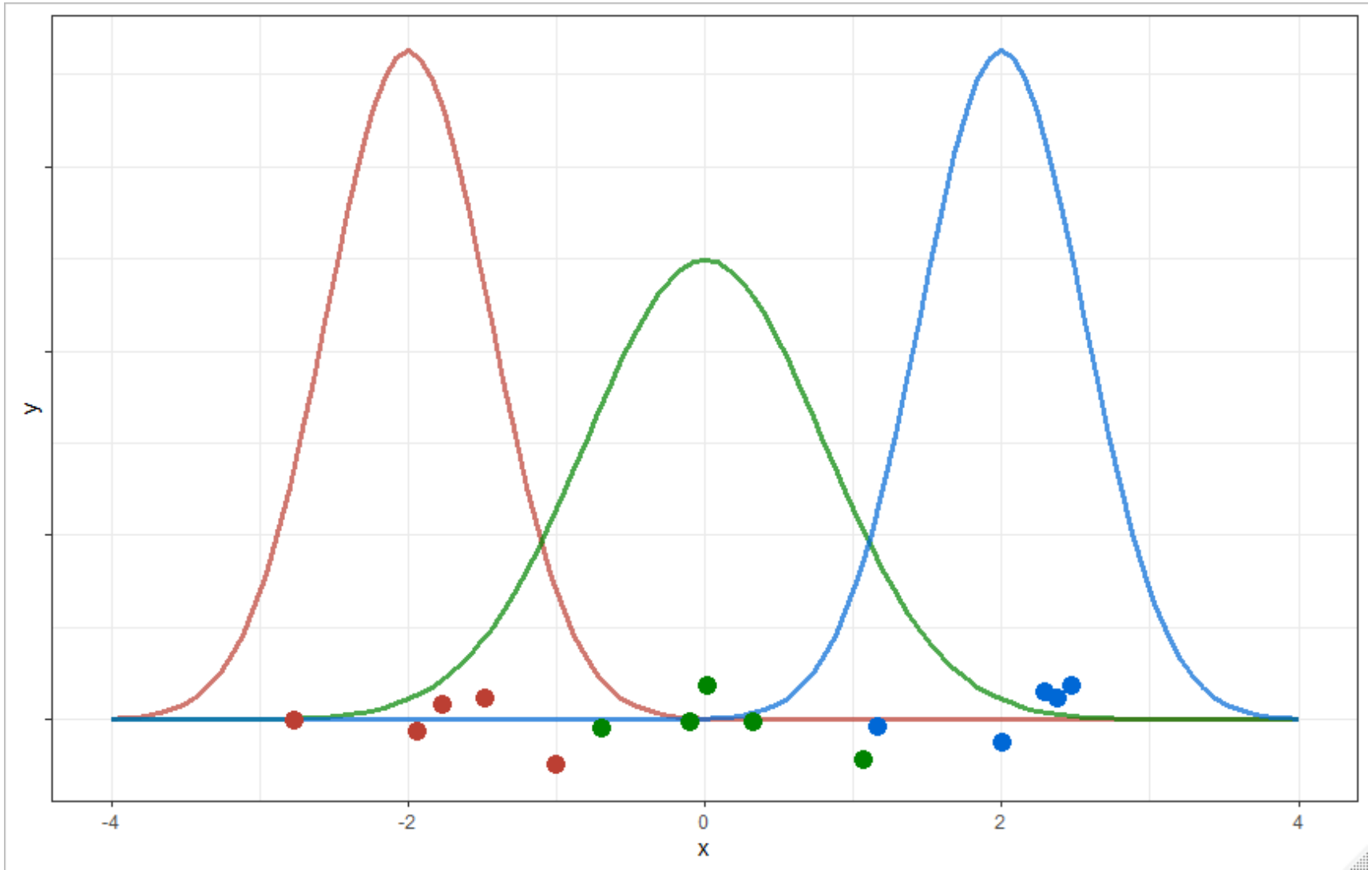
Cluster analysis groups the observations into "interesting" **clusters**

- The observations within each cluster should be like each other.
- The observations across clusters should not be like each other.
- What defines similarly?
- Let's use a few simple examples to find out.

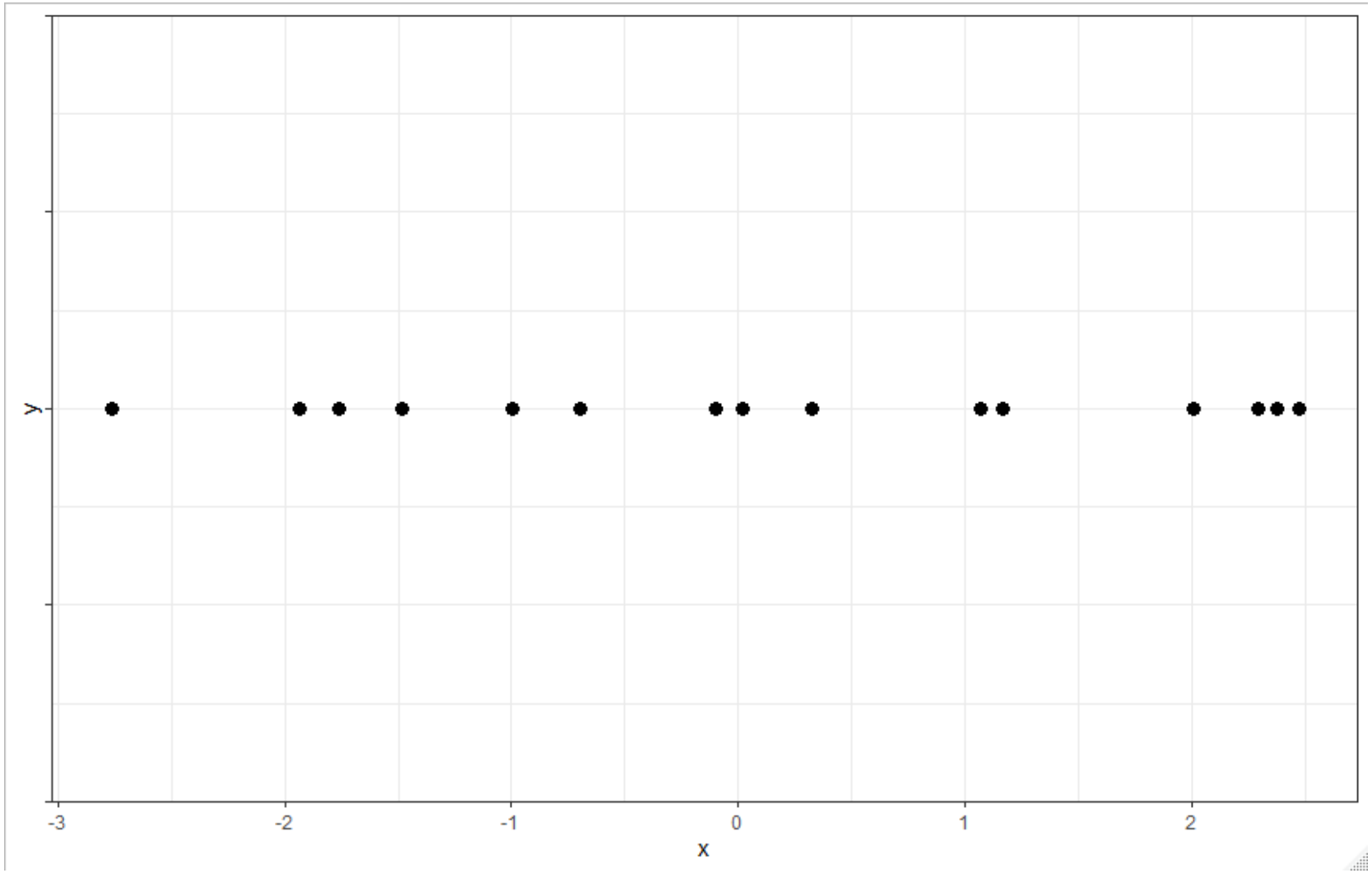
Let’s start out with a simple 1D example

- Generate 5 random draws from 3 different Gaussian distributions.
- Means of -2, 0, and +2 and standard deviations of 0.55, 0.8, and 0.55, respectively.
- Can we cluster the 15 observations into 3 distinct groups?
- Example adapted from StatQuest K-means clustering example https://www.youtube.com/watch?v=4b5d3muPQmA

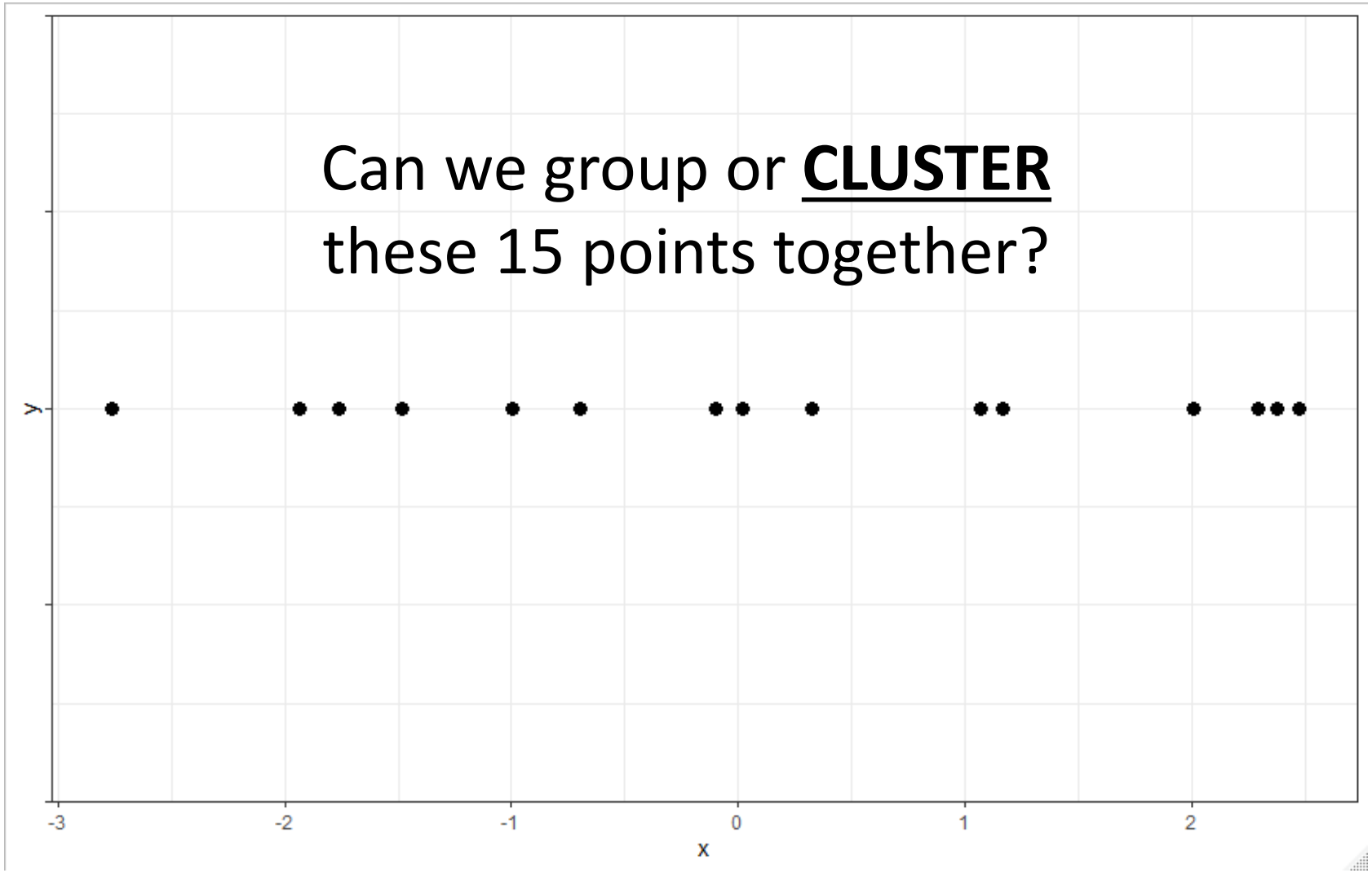
Random draws color coded by their respective Gaussians



Let’s pretend we don’t know which Gaussian the observations are associated with



Let’s pretend we don’t know which Gaussian the observations are associated with



To cluster the observations together we must consider the following

How many clusters?

How do we define a cluster?

To cluster the observations together we must consider the following

How many clusters?

In KMeans clustering, we must specify the number of clusters **upfront**.

KMeans does **NOT** pick the OPTIMAL number of clusters!

How do we define a cluster?

To cluster the observations together we must consider the following

How many clusters?

In KMeans clustering, we must specify the number of clusters **upfront**.

KMeans does **NOT** pick the OPTIMAL number of clusters!

How do we define a cluster?

A “good” cluster is one that has a small **within cluster variation**.

Real question then is, what defines **within cluster variation**?

Data organization required by KMeans

- The variables are contained in an array $\mathbf{X}$ with N rows and D columns.
- The n-th row and d-th column is: $x_{n,d}$
- Think of $\mathbf{X}$ as a 2D NumPy array, `X`. Thus, $x_{n,d}$ corresponds to slicing the array as: `X[n, d]`.
- KMeans wants WIDE-FORMAT data!!!!

Similarity is based on DISTANCE

- The closer two data points are…the MORE similar they are!
- The further two data points are…the LESS similar they are!
- How do we calculate distance? SUBTRACT!!
  - The distance between the n-th and m-th rows for the d-th column:

```
np.abs( X[ n, d ] - X[ m, d ] )
```

What if we have TWO variables?

- We need to consider BOTH variables!
- But…we should NOT just simply subtract…we SUM the SQUARED DIFFERENCES!
- For example, the DISTANCE between the n-th and m-th rows BASED on the d-th and k-th columns:

```
(X[ n, d ] - X[ m, d ])**2 + (X[ n, k ] - X[ m, k ])**2
```

Euclidean distance is the SQUARE ROOT of the SUM of SQUARED DIFFERENCES

```
np.sqrt( (X[ n, d ] – X[ m, d ])**2 + (X[ n, k ] – X[ m, k ])**2 )
```

Euclidean distance is the SQUARE ROOT of the SUM of SQUARED DIFFERENCES

```
np.sqrt( (X[ n, d ] - X[ m, d ])**2 + (X[ n, k ] - X[ m, k ])**2 )
```

The **Squared Euclidean distance** does NOT apply the SQUARE ROOT:

```
(X[ n, d ] - X[ m, d ])**2 + (X[ n, k ] - X[ m, k ])**2
```

In math notation, the Squared Euclidean Distance between TWO rows based on D columns

$$\text{distance}(n, m) = \sum_{d=0}^{D-1} \left(\left(x_{n,d} - x_{m,d} \right)^2 \right)$$

Why does this matter? We are calculating the SQUARED DISTANCE based on ALL variables!

We MUST be able to calculate the difference!

We can ONLY use NUMERIC columns!!!

Other distance metrics exist besides just the (squared) Euclidean distance

- Some even work with non-numeric columns!

- However, we will focus on the Euclidean distance in this course.

Why does this matter for KMeans?

- KMeans calculates the WITHIN CLUSTER variation as the SUM of the squared Euclidean distances between all observations within a cluster!
- This quantity is known as the **within sum of squares** for a cluster!
- Clusters with many SIMILAR observations have very SMALL within sum of squares values!
- Clusters with many DIFFERENT observations have very LARGE within sum of squares values.

Why does this matter for KMeans?

- An important METRIC for KMeans is the **Total Within Sum of Squares**.

- The Total Within Sum of Squares is calculated by SUMMING (adding) the within sum of squares across ALL clusters!

The objective of KMeans clustering is to **minimize** the total within sum of squares

- Minimizing the total within sum of squares creates clusters with observations that are VERY CLOSE together!

- Within sum of squares is VERY SIMILAR to VARIANCE!

- KMeans tries to group observations together to minimize the VARIANCE within each cluster!!

How do we minimize the total within sum of squares?

- Are there parameters or coefficients to learn in this problem?

How do we minimize the total within sum of squares?

- Are there parameters or coefficients to learn in this problem?

- NO!

- We minimize the total within sum of squares by **ASSIGING** each observation to a cluster!

Partitioning or grouping the N observations into K clusters is quite hard…

- There are about K^N ways to do this…

- For our simple 1D example with $K = 3$ and $N = 15$, there are 3^{15} or about 14 million combinations!!!

We cannot try out all possible combinations…

- Instead, let's iterate!
  - Start from a **RANDOM** initial guess.
  - Assign observations based on PROXIMITY to the nearest cluster center.
  - Iterate until convergence.

- Solution is **not** the global optimal cluster assignment but yields a pretty good result.

- Algorithm is simple to setup and execute.

KMeans clustering algorithm

- Specify the number of clusters, K.
- Randomly assign each observation to one of the K clusters.
- Iterate until cluster assignments stop changing:
  - Compute the **centroid** for each cluster.
  - Assign observations to the closest cluster, based on the Euclidean distance to the cluster centroid.

The algorithm is guaranteed to decrease the total within sum-of-squares

- Assigning each observation to the closest cluster center reduces the total within sum of squares!

- Assigning each observation to the closest cluster center therefore reduces the VARIANCE within each cluster!

The algorithm cannot identify the global optimum…the solution is a local optimum

- What are the practical implications of that?

- Remember, the algorithm starts from a random initial guess…

The algorithm cannot identify the global optimum…the solution is a local optimum

- What are the practical implications of that?

- The result may depend on the initial guess!!

- Try out **MANY** different random initial guesses and compare the total within sum of squares.

Let’s try out KMeans clustering on the simple 1D example

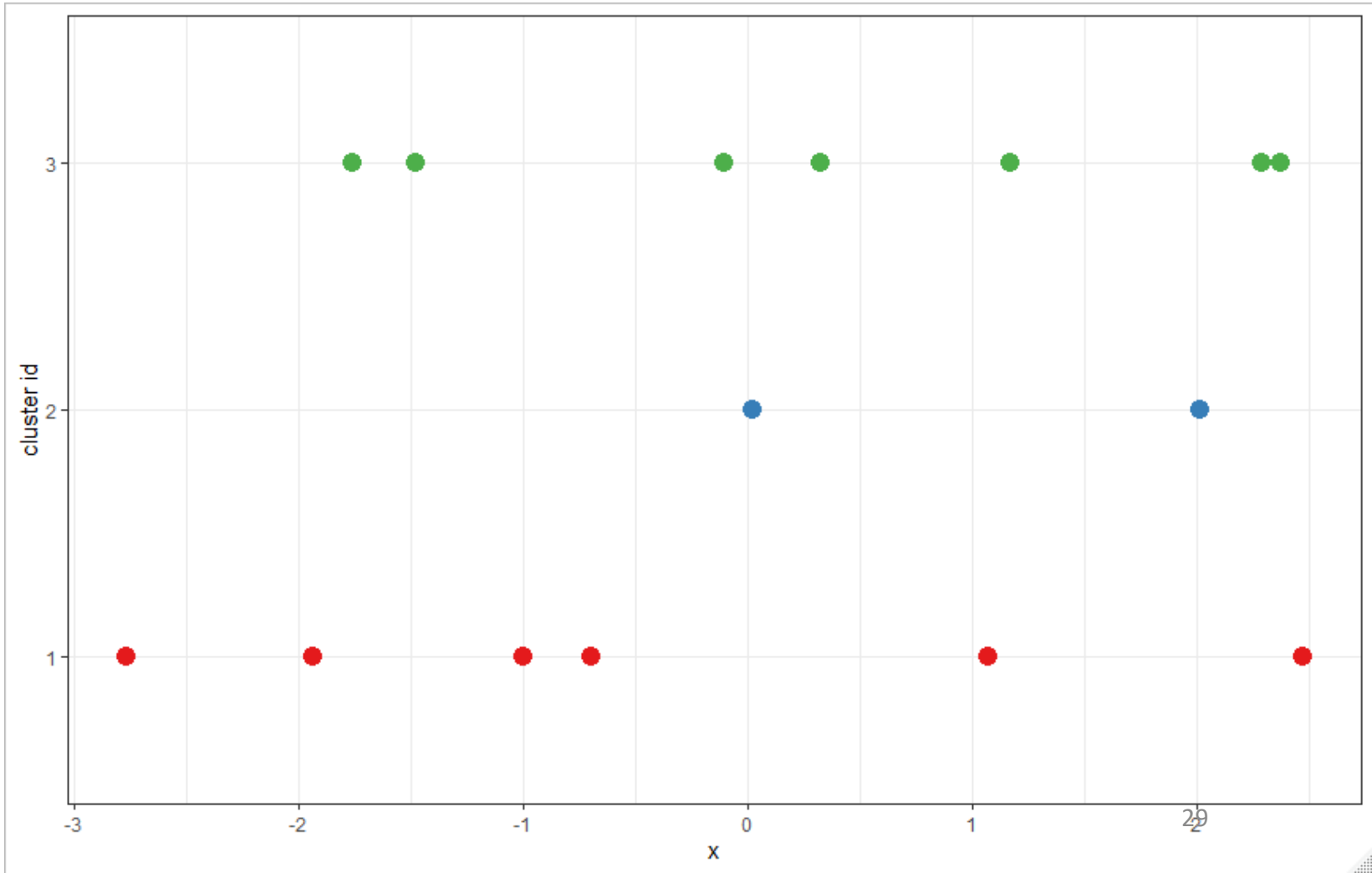
- Try out 3 clusters, we know that’s the right answer in this toy problem.

- Let’s see if the KMeans algorithm can identify the correct 3 clusters.

Randomly assign each observation to one of three possible clusters

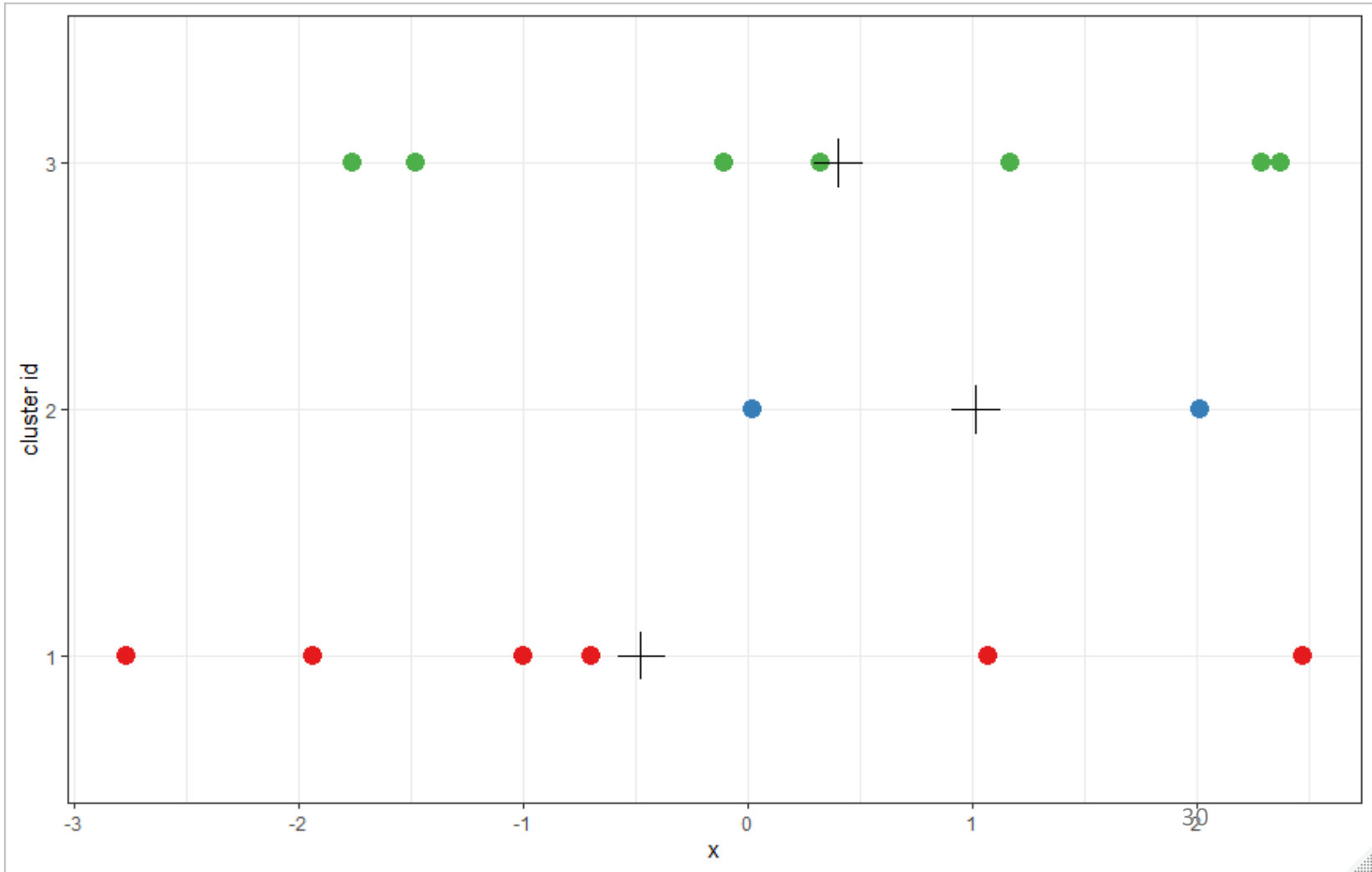
Color denotes the random cluster assignment.

Clusters are separated vertically to make it easier to see each point within each cluster.



Calculate the centroid associated with each cluster

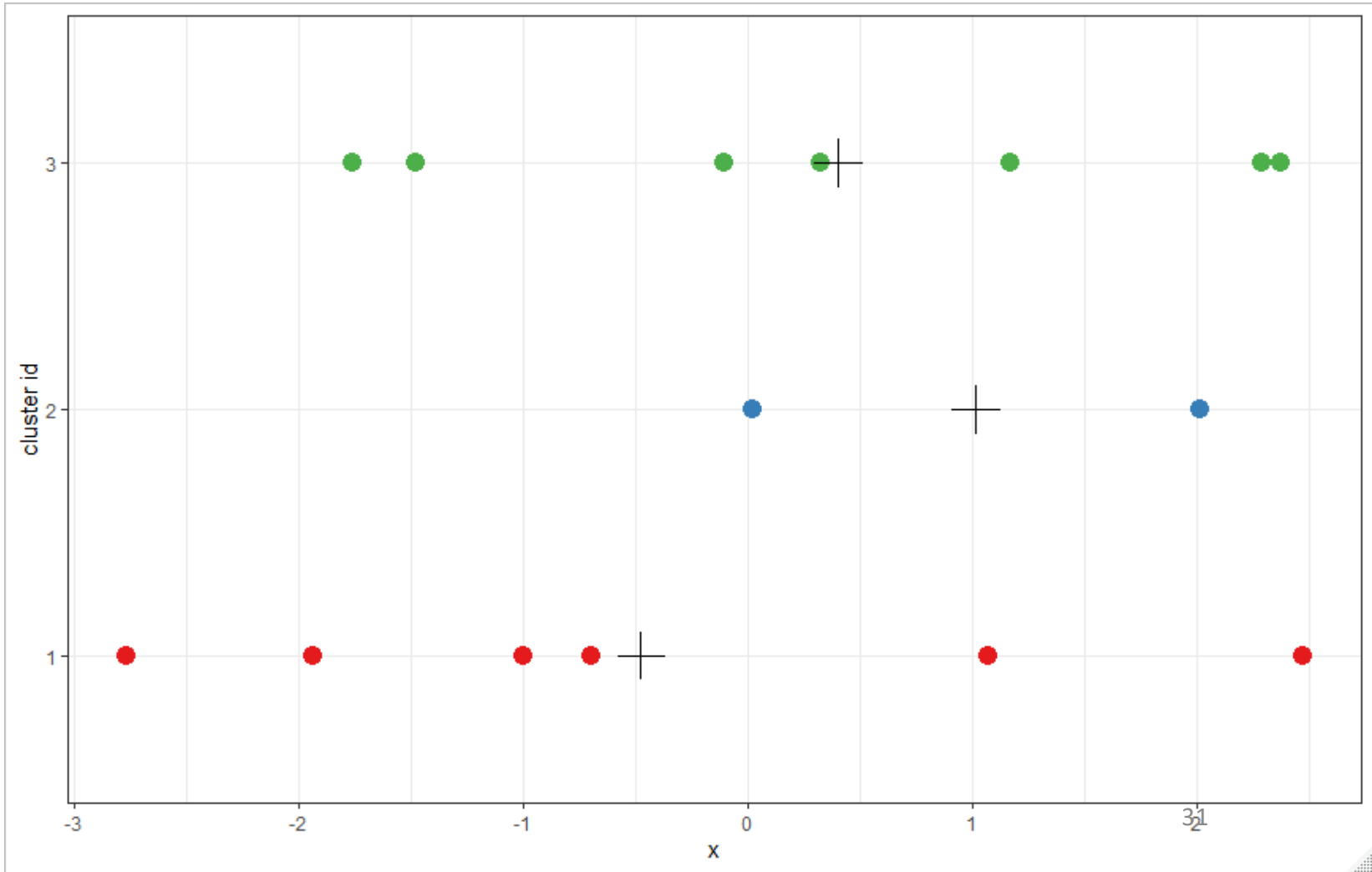
Black + symbols denote the average or centroid associated with each cluster.



Reassign observations to clusters based on the distance to the cluster centroids

The vertical “distance” in the figure is fake.

The horizontal distance is the only one that matters in this 1D example.

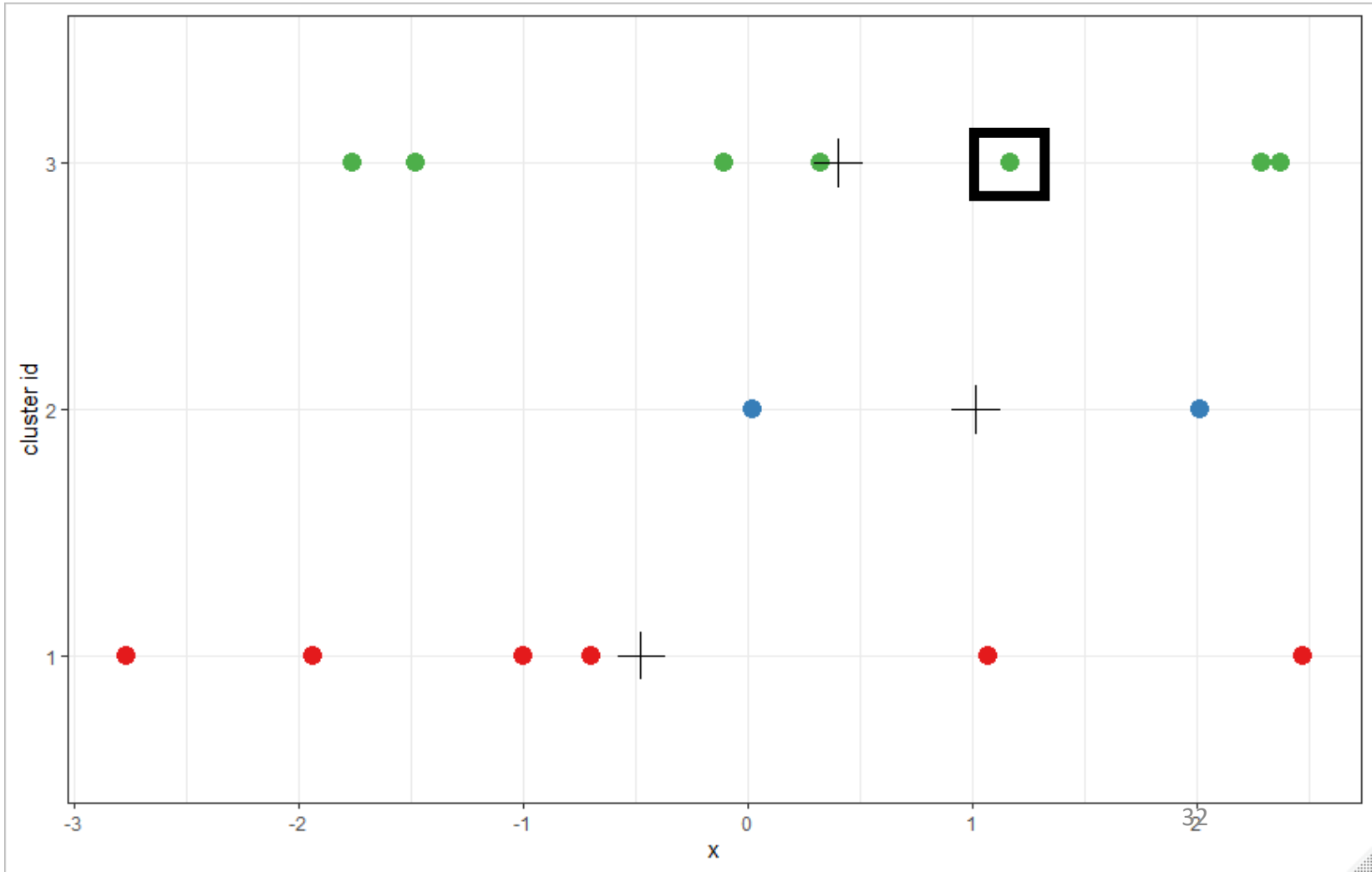


Reassign observations to clusters based on the distance to the cluster centroids

Consider the point marked by the black square outline.

Which cluster should it be assigned to?

Calculate the distance between it and the 3 cluster centroids.

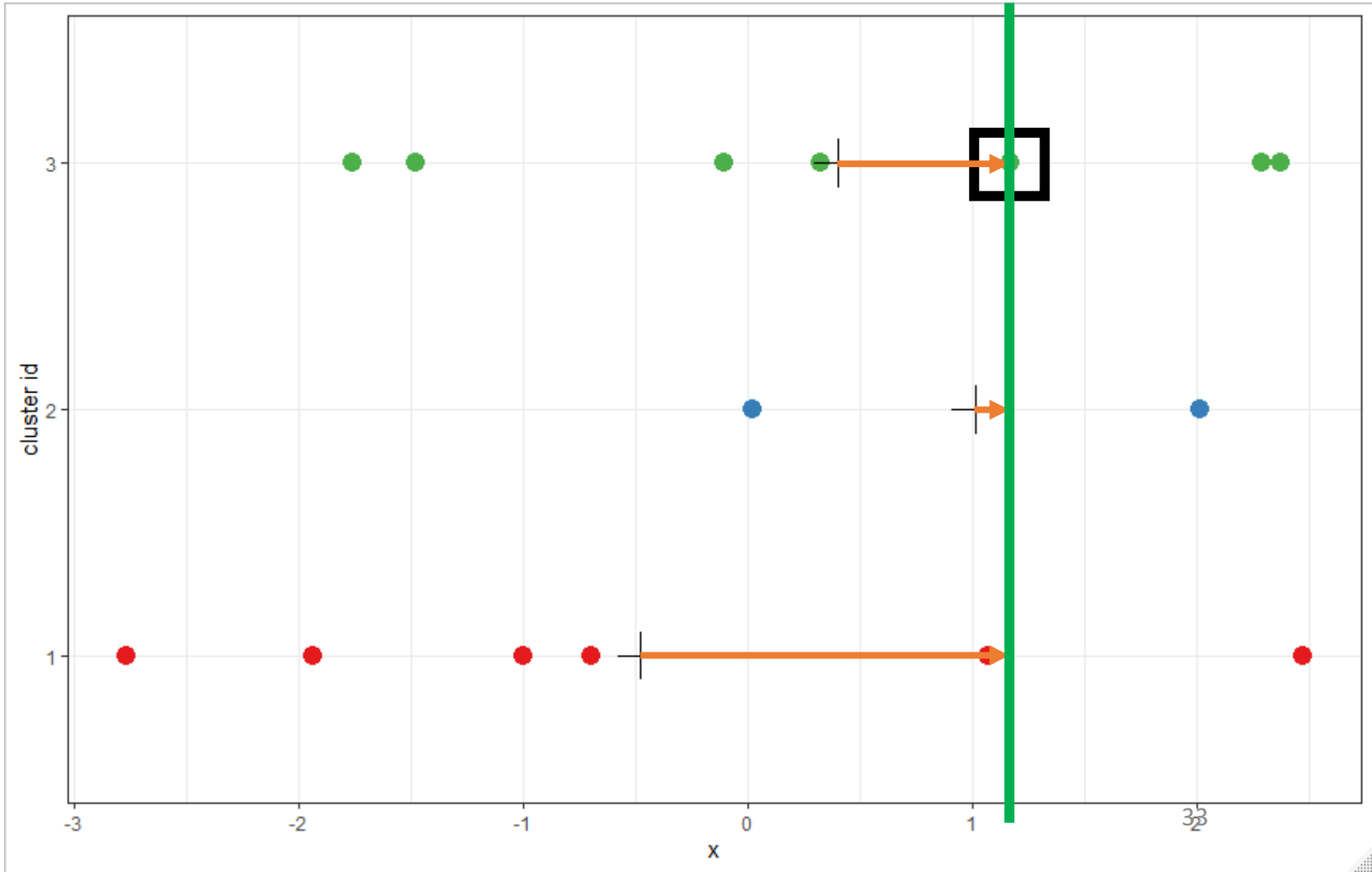


Reassign observations to clusters based on the distance to the cluster centroids

Consider the point marked by the black square outline.

Which cluster should it be assigned to?

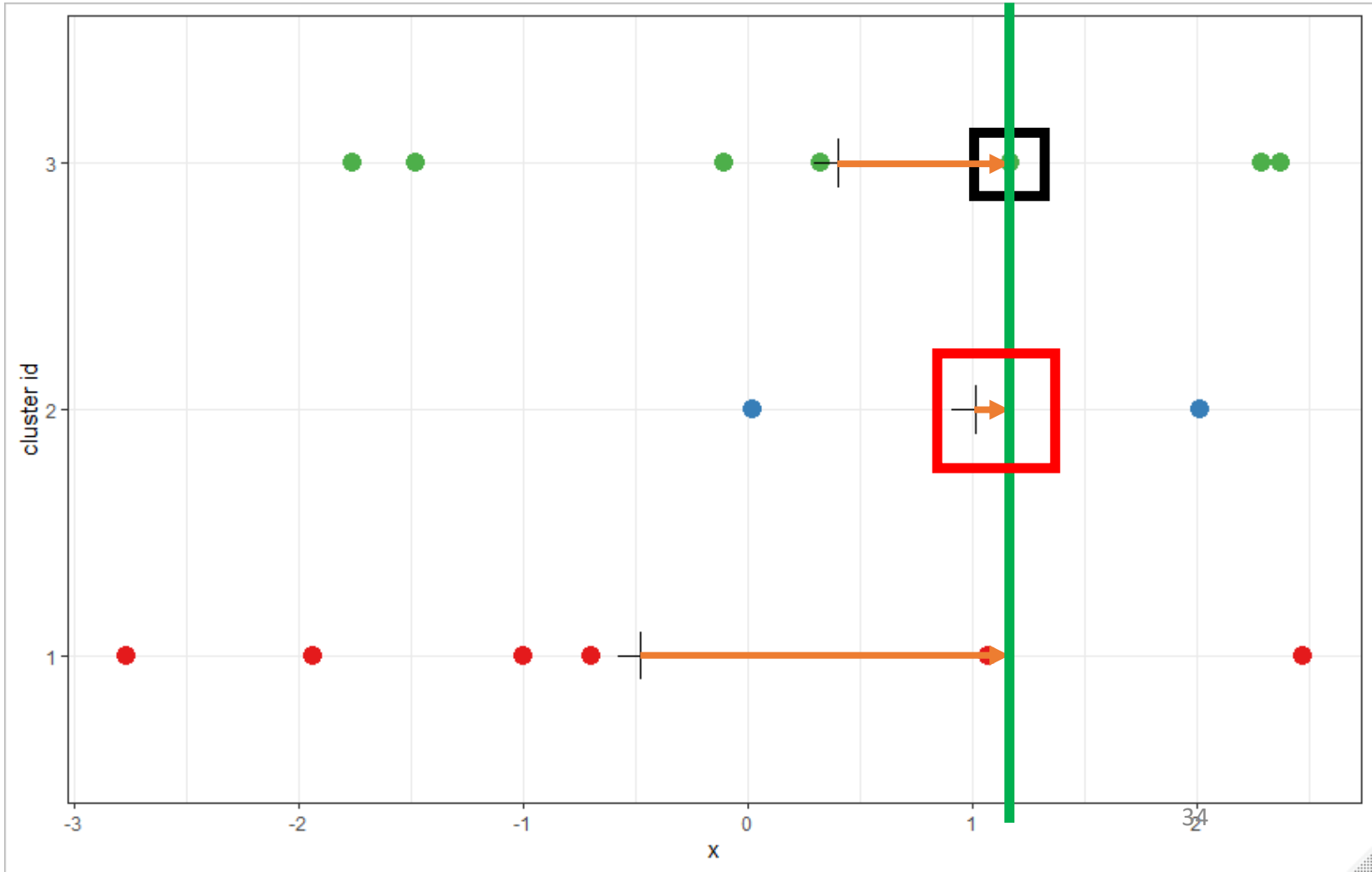
Calculate the distance between it and the 3 cluster centroids.



Reassign observations to clusters based on the distance to the cluster centroids

The point is closest to the centroid from cluster 2.

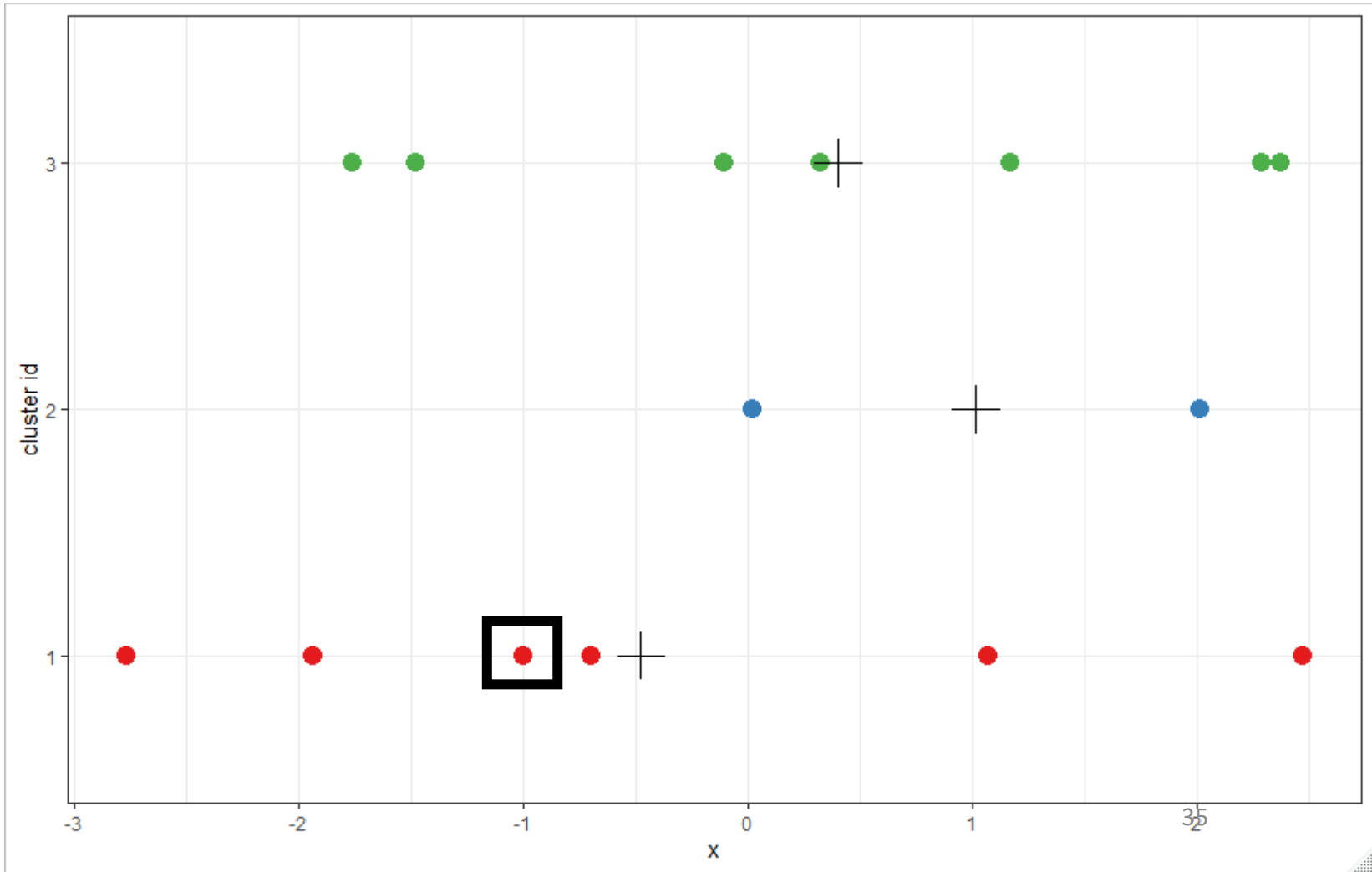
Reassign point from cluster 3 to cluster 2.



Reassign observations to clusters based on the distance to the cluster centroids

Repeat for another point marked by the black square.

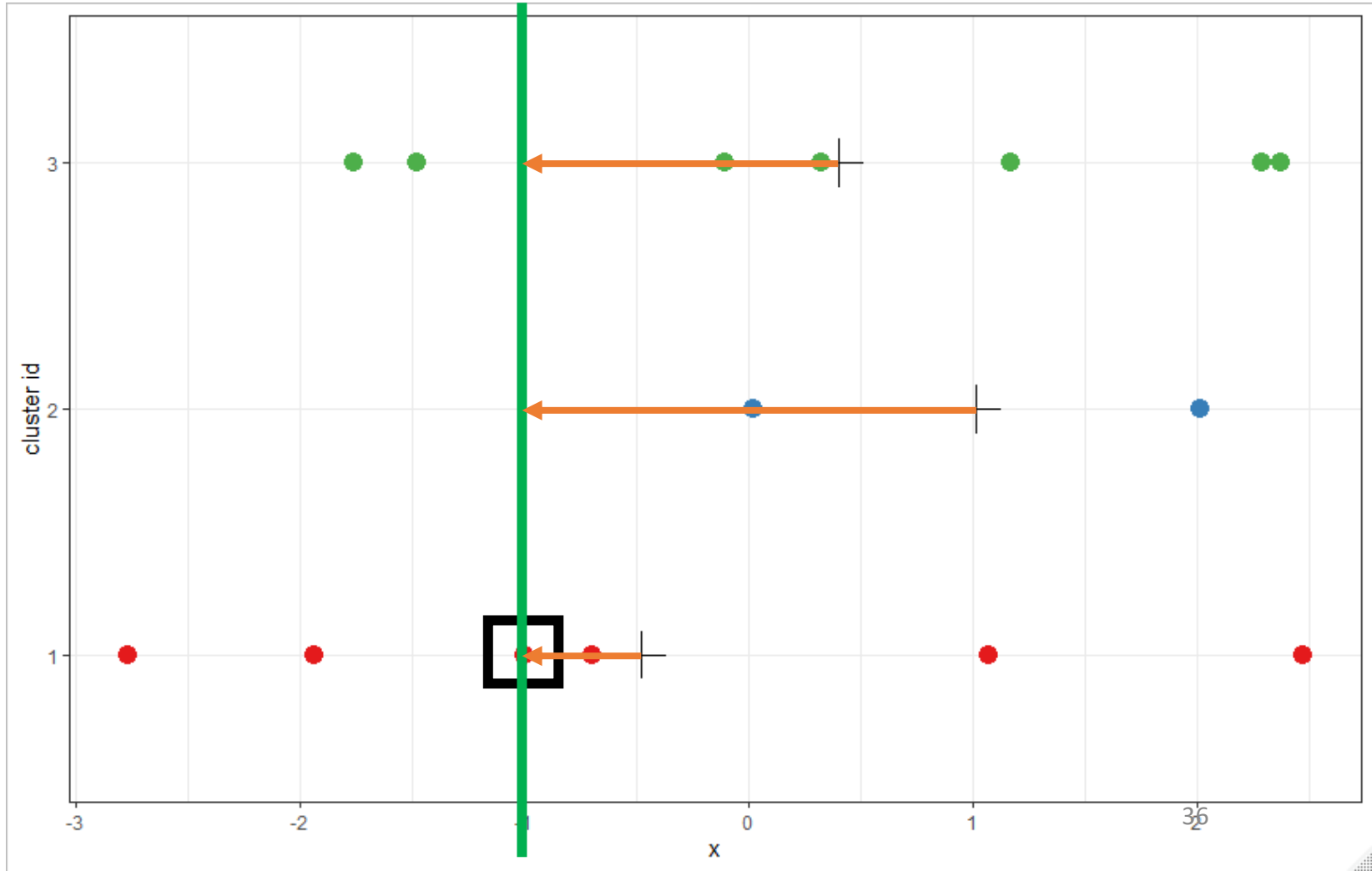
Which cluster should it be assigned to?



Reassign observations to clusters based on the distance to the cluster centroids

Repeat for another point marked by the black square.

Closest cluster is cluster 1.

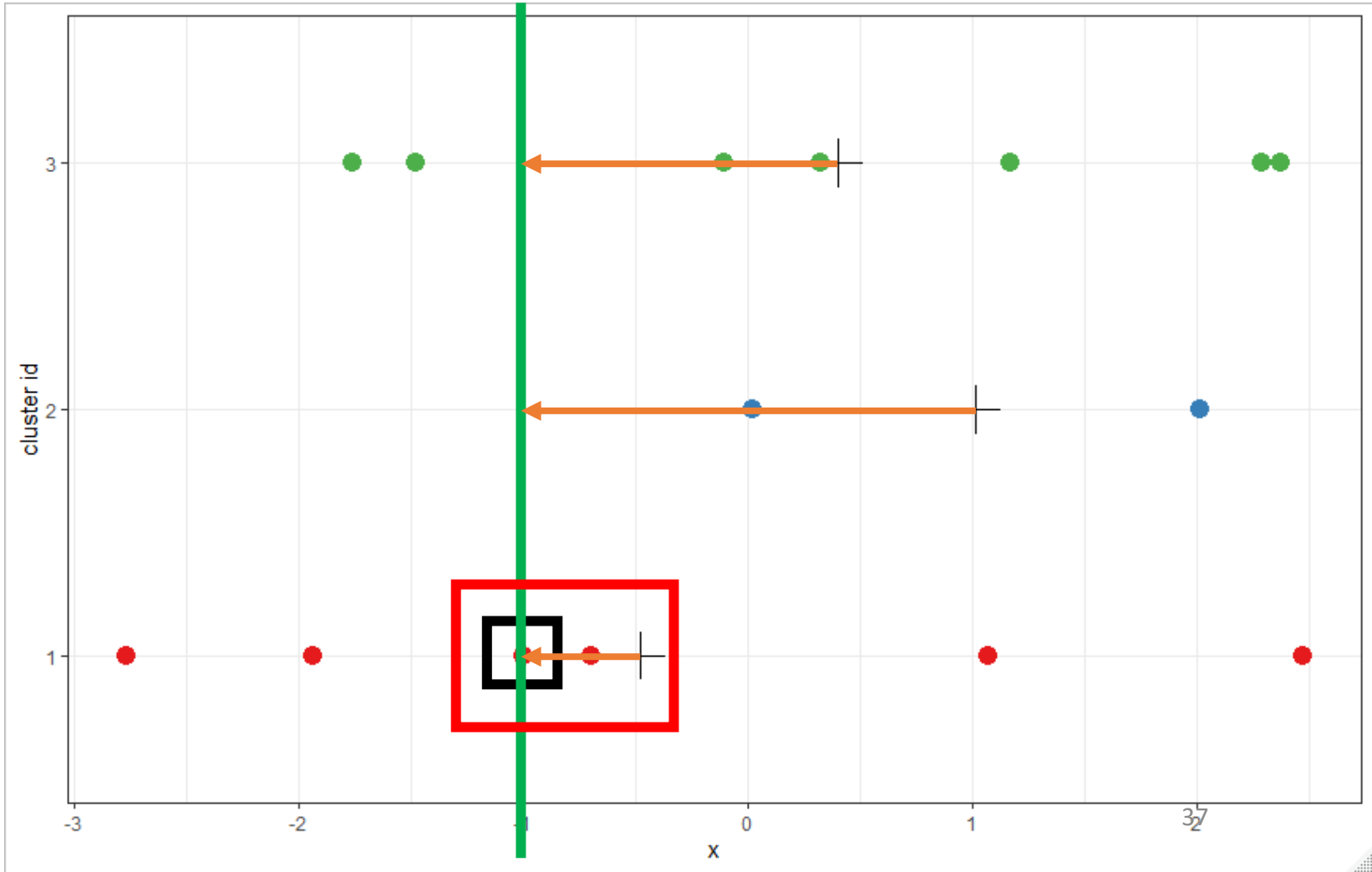


Reassign observations to clusters based on the distance to the cluster centroids

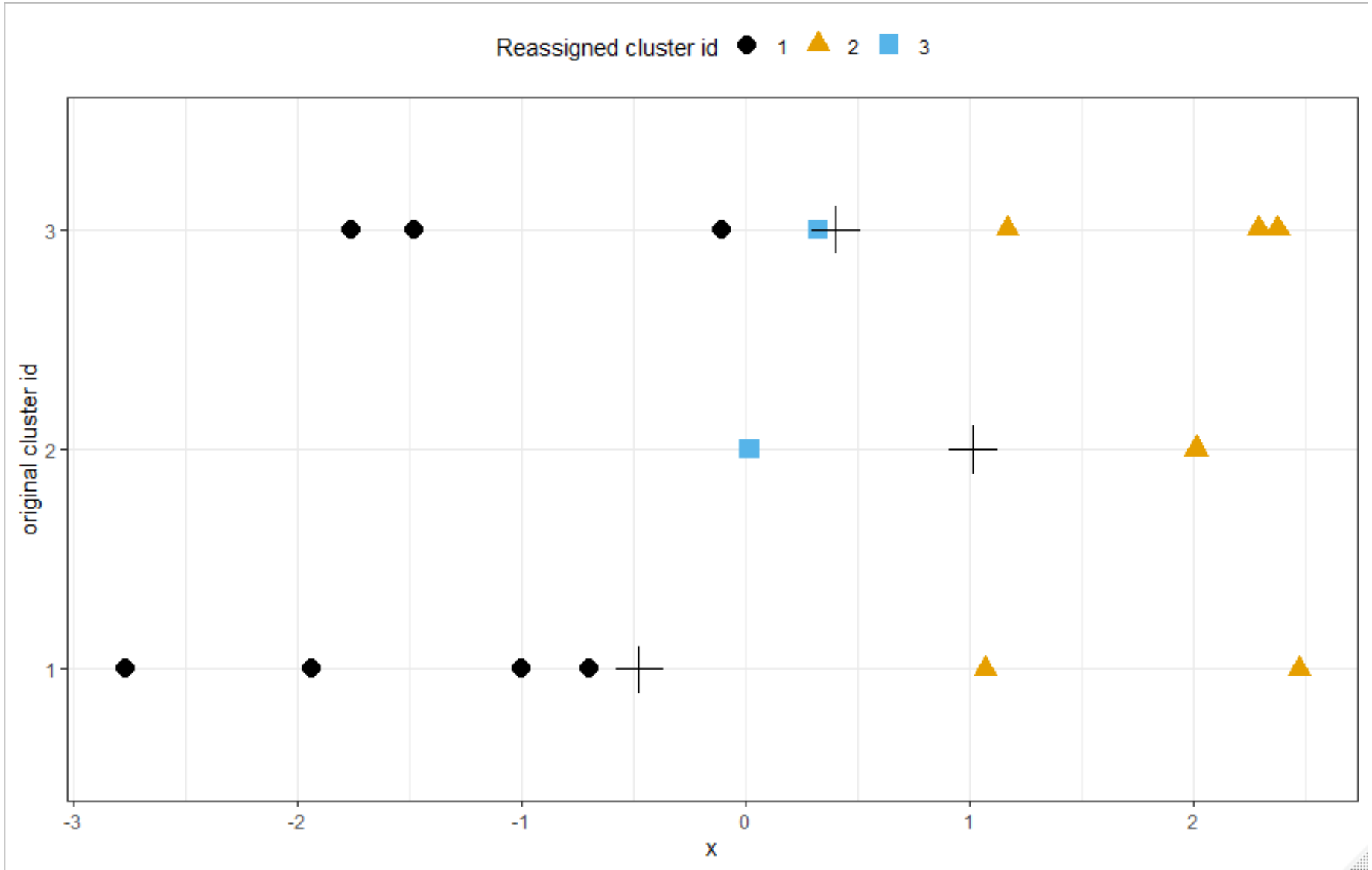
Repeat for another point marked by the black square.

Closest cluster is cluster 1.

The point remains assigned to cluster 1.

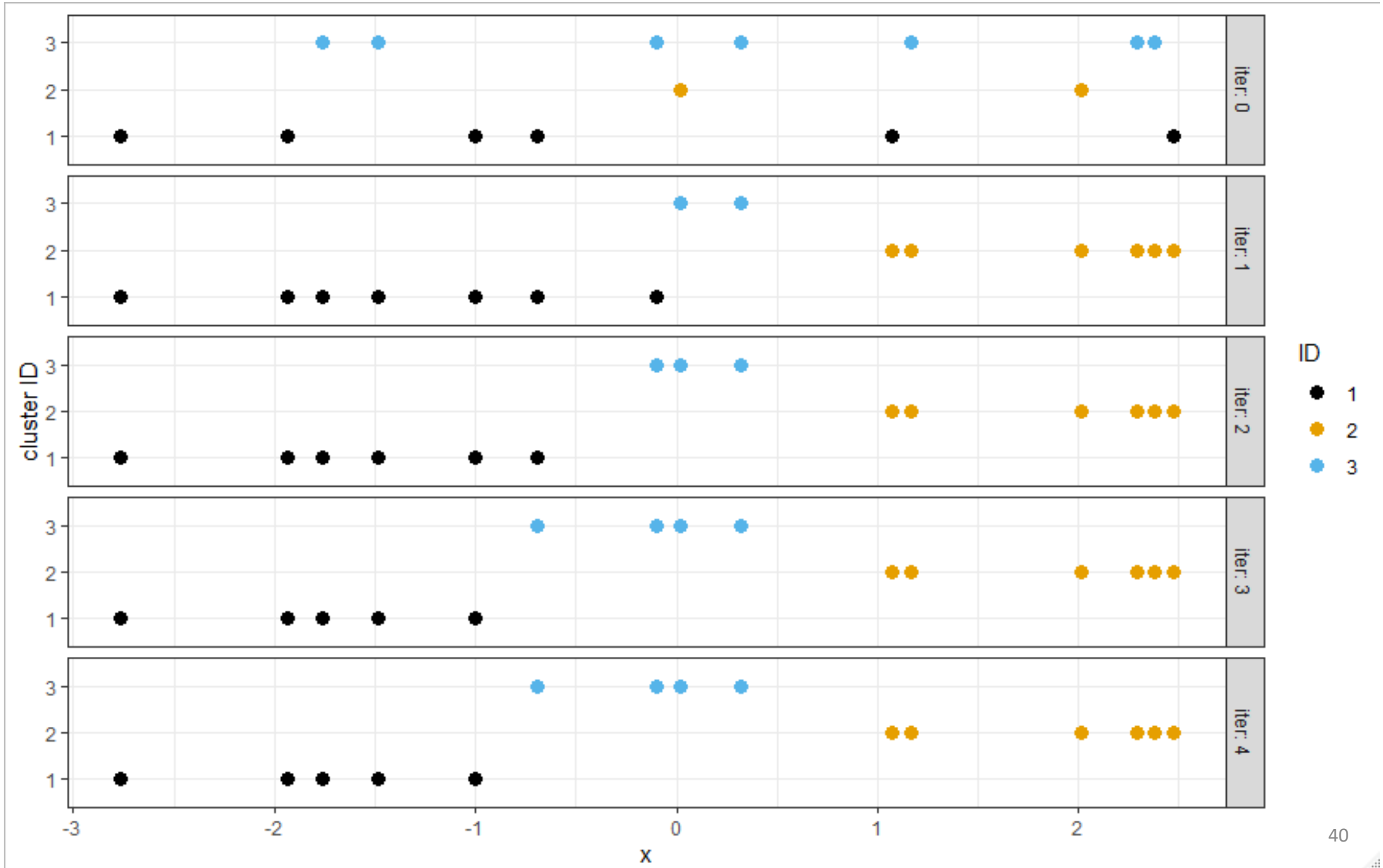


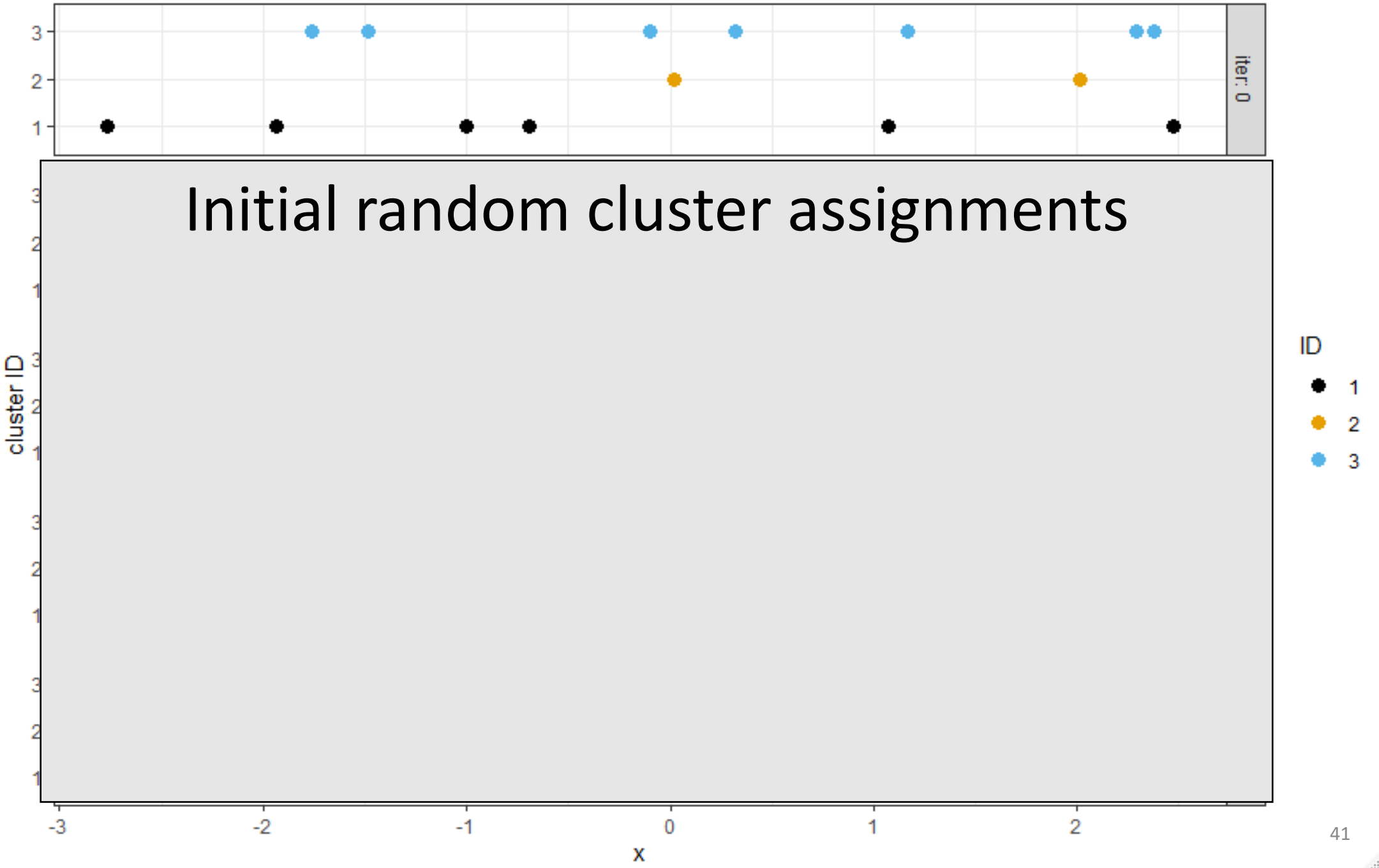
Repeat for all observations.

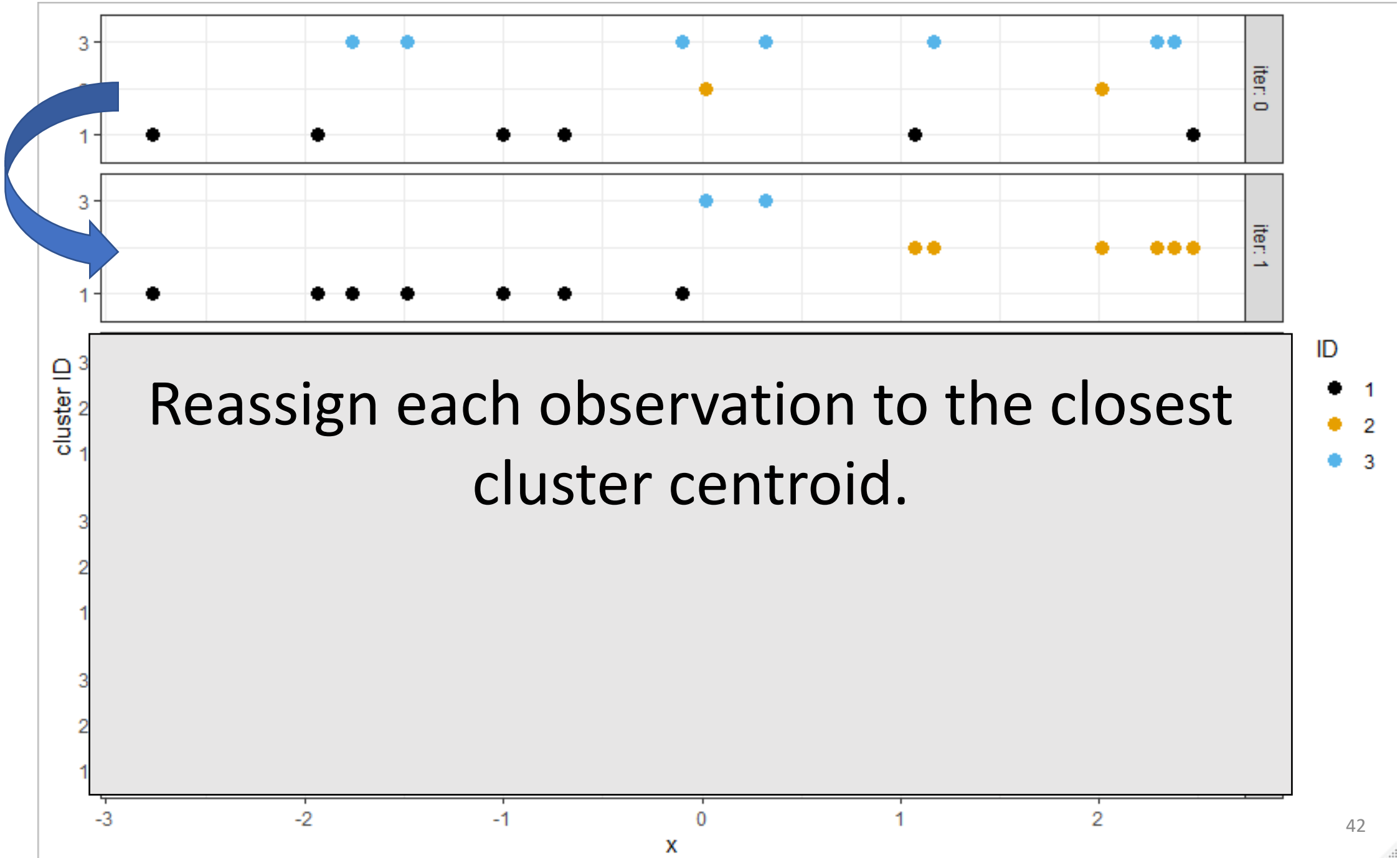


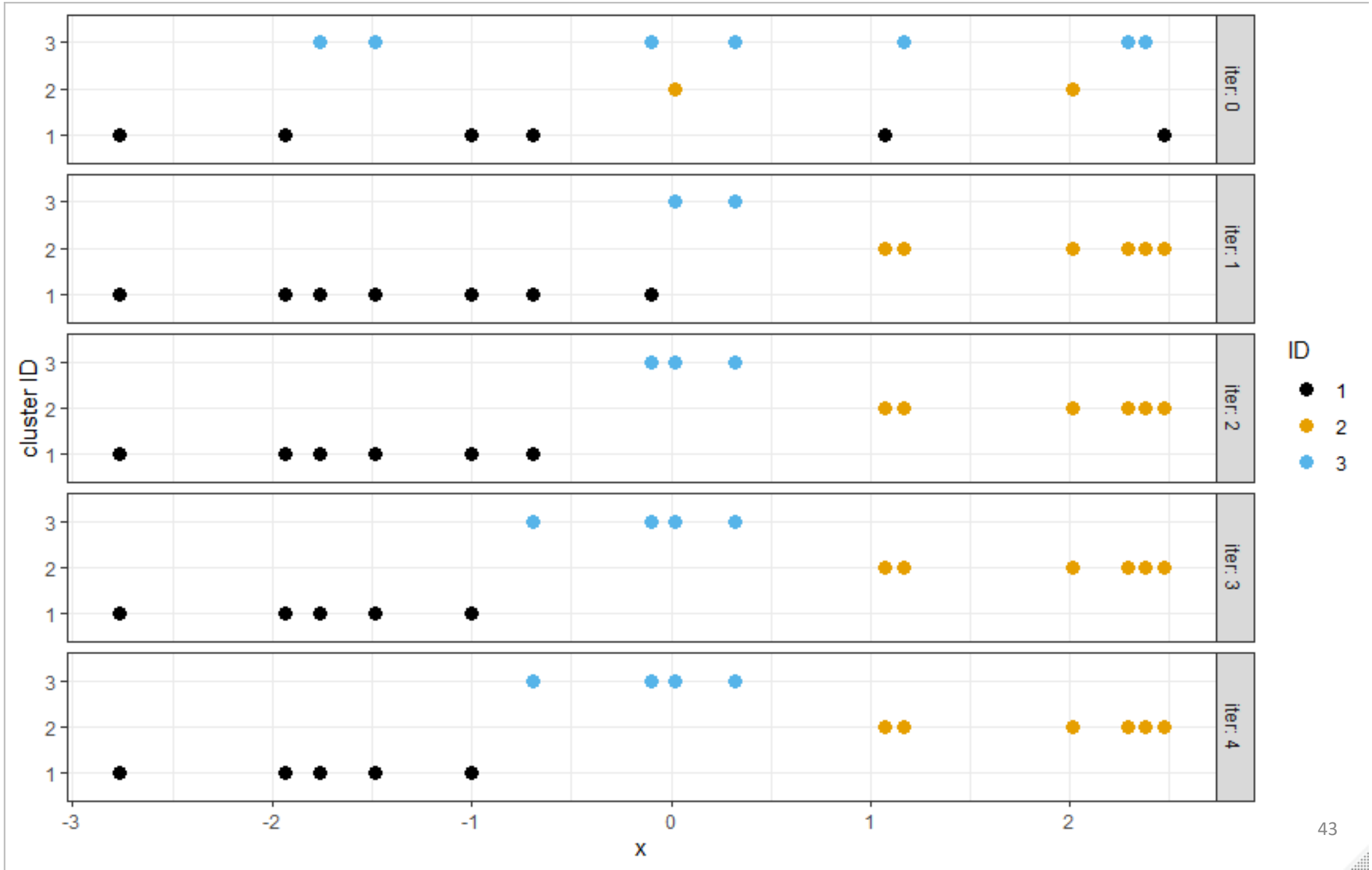
This was just the first iteration…

- Repeat!!
- The cluster reassignments act as the starting guess for the next iteration.
- Recalculate the cluster centroids and reassign based on the new cluster centroids.

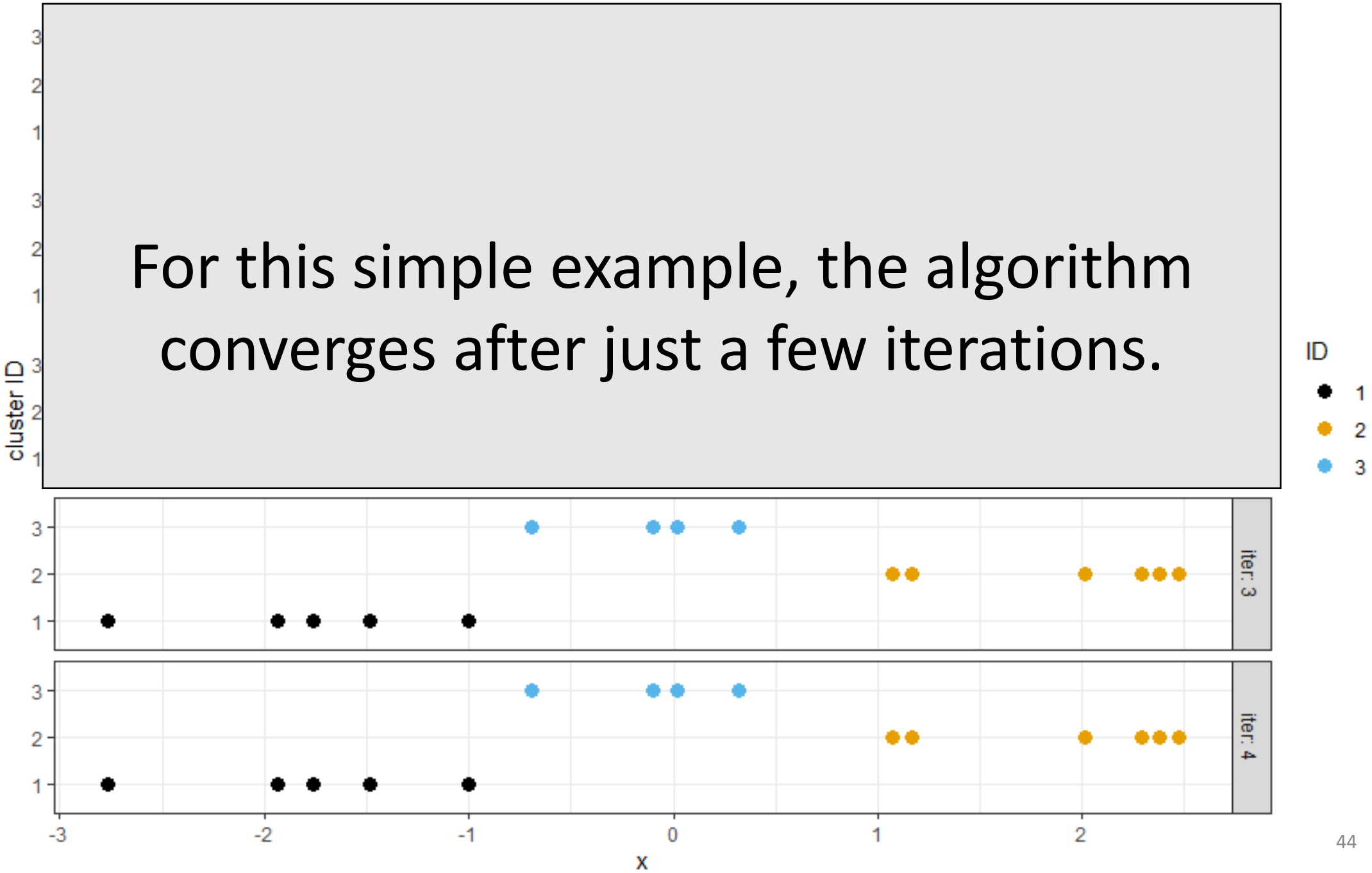

iter: 0
iter: 1
iter: 2
iter: 3
iter: 4
cluster ID
1
2
3
x
-3
-2
-1
0
1
2
ID
1
2
3


iter: 0
Initial random cluster assignments
cluster ID
x
ID
1
2
3


iter: 0
iter: 1
cluster ID
x
ID
1
2
3
Reassign each observation to the closest cluster centroid.


iter: 0
iter: 1
iter: 2
iter: 3
iter: 4
cluster ID
x
ID
1
2
3

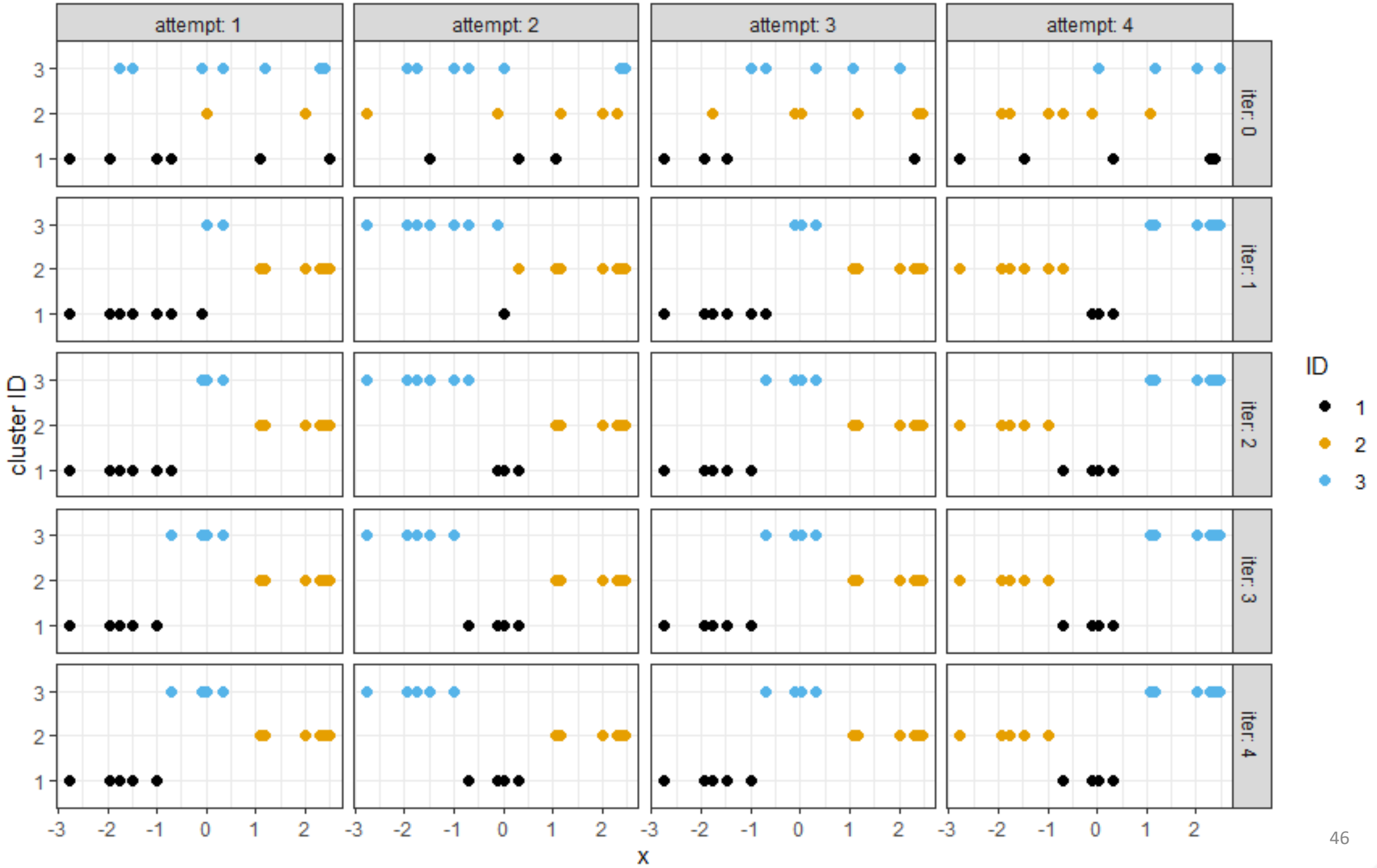
For this simple example, the algorithm converges after just a few iterations.

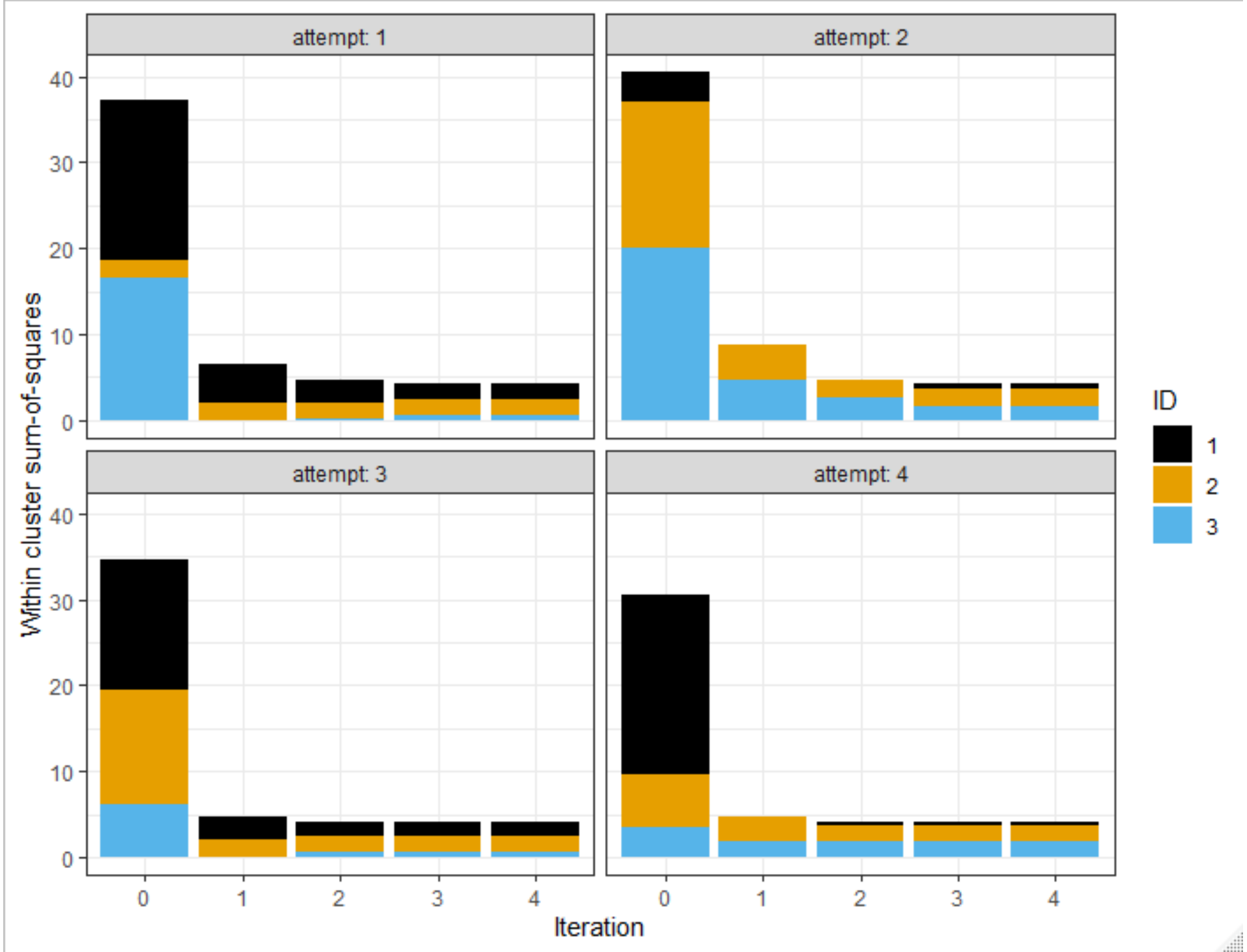


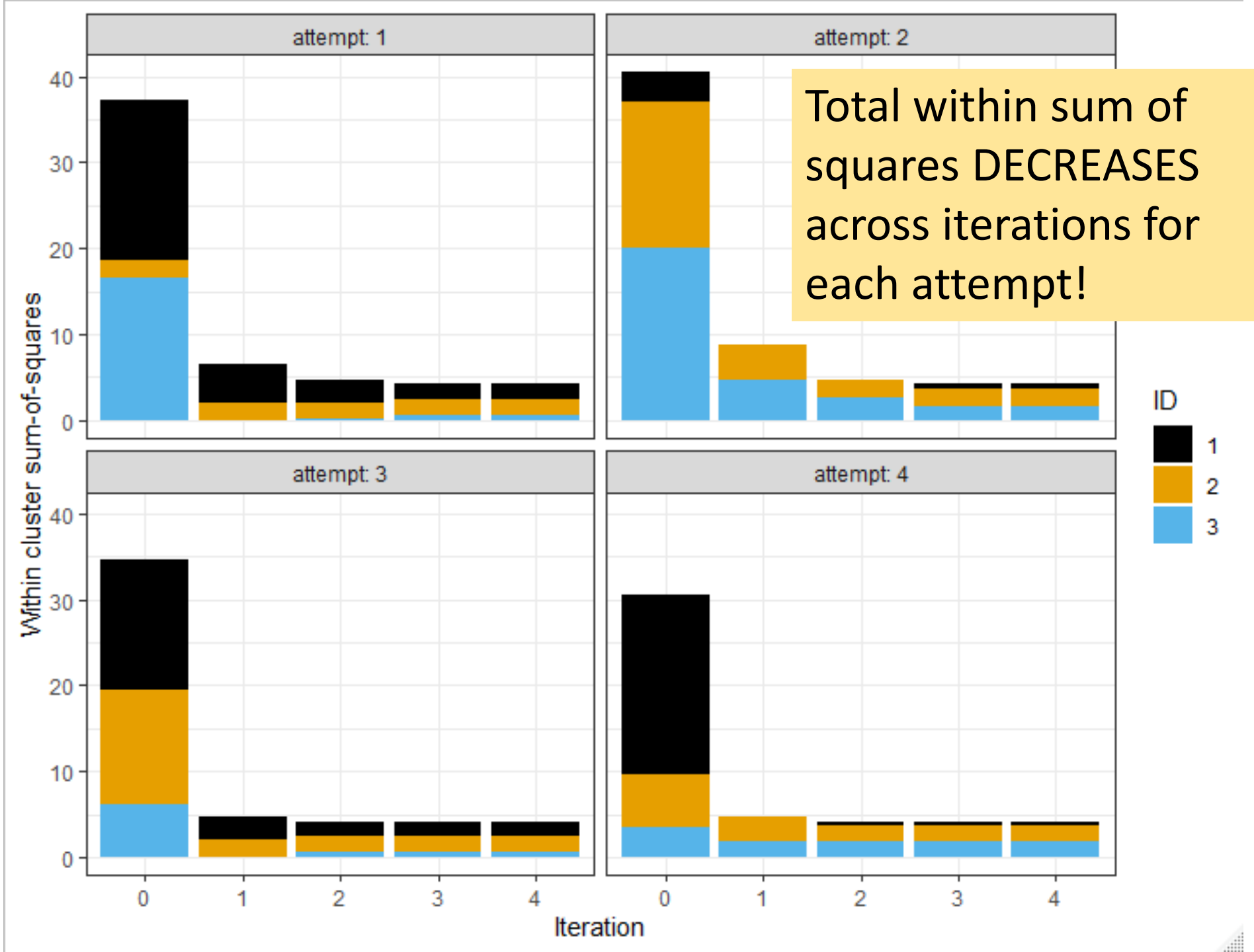
Although the algorithm converged, this was from ONE random initial guess

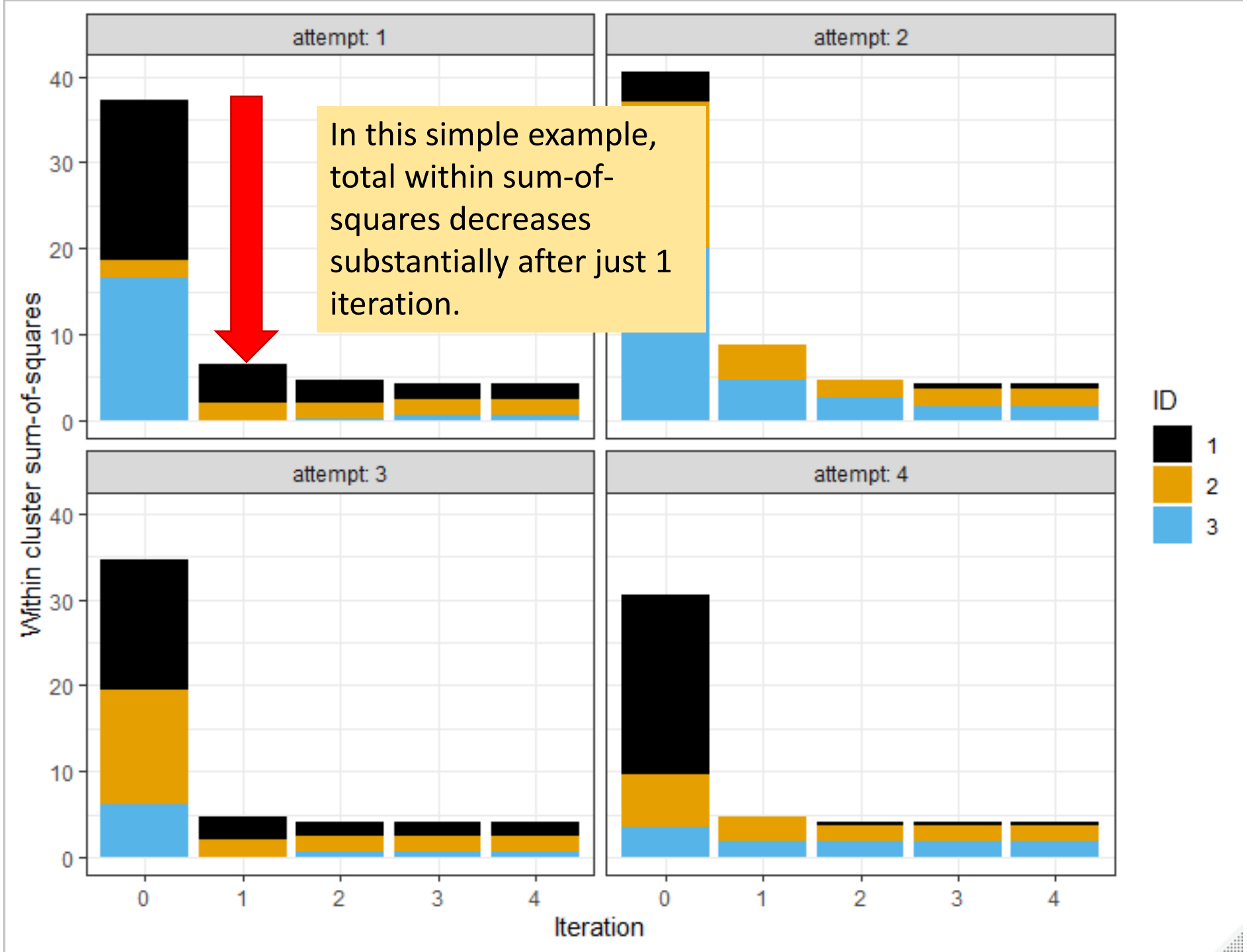
- We need to try out multiple random initial cluster assignments.

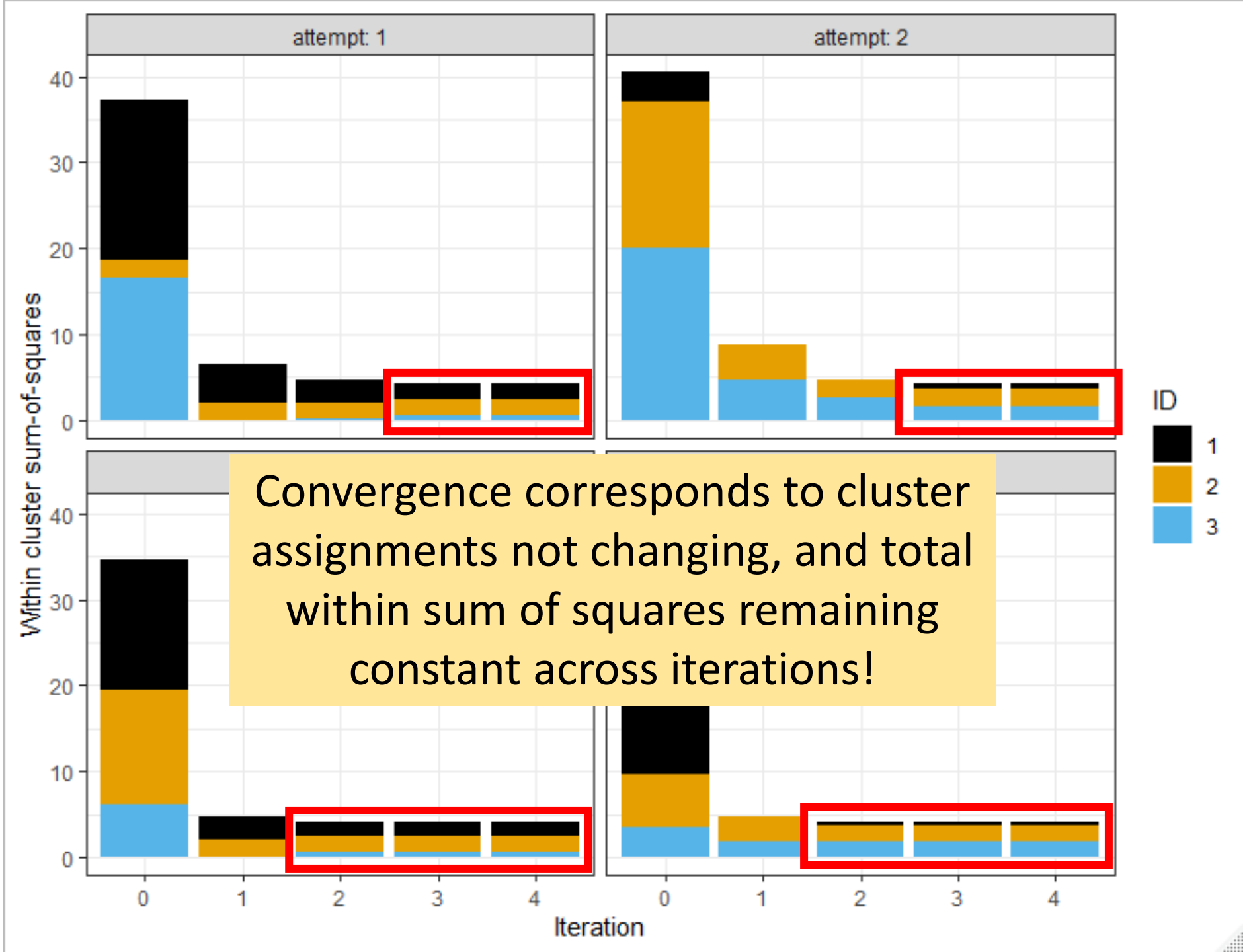
- Compare the results across the different random "restarts".


attempt: 1
attempt: 2
attempt: 3
attempt: 4
iter: 0
iter: 1
iter: 2
iter: 3
iter: 4
cluster ID
x
ID
1
2
3


attempt: 1
attempt: 2
attempt: 3
attempt: 4
Within cluster sum-of-squares
Iteration
ID
1
2
3
0
10
20
30
40
0
1
2
3
4


attempt: 1
attempt: 2
attempt: 3
attempt: 4
Within cluster sum-of-squares
Iteration
ID
1
2
3
Total within sum of squares DECREASES across iterations for each attempt!


attempt: 1
attempt: 2
attempt: 3
attempt: 4
In this simple example, total within sum-of-squares decreases substantially after just 1 iteration.
Within cluster sum-of-squares
Iteration
ID
1
2
3
0
10
20
30
40
0
1
2
3
4



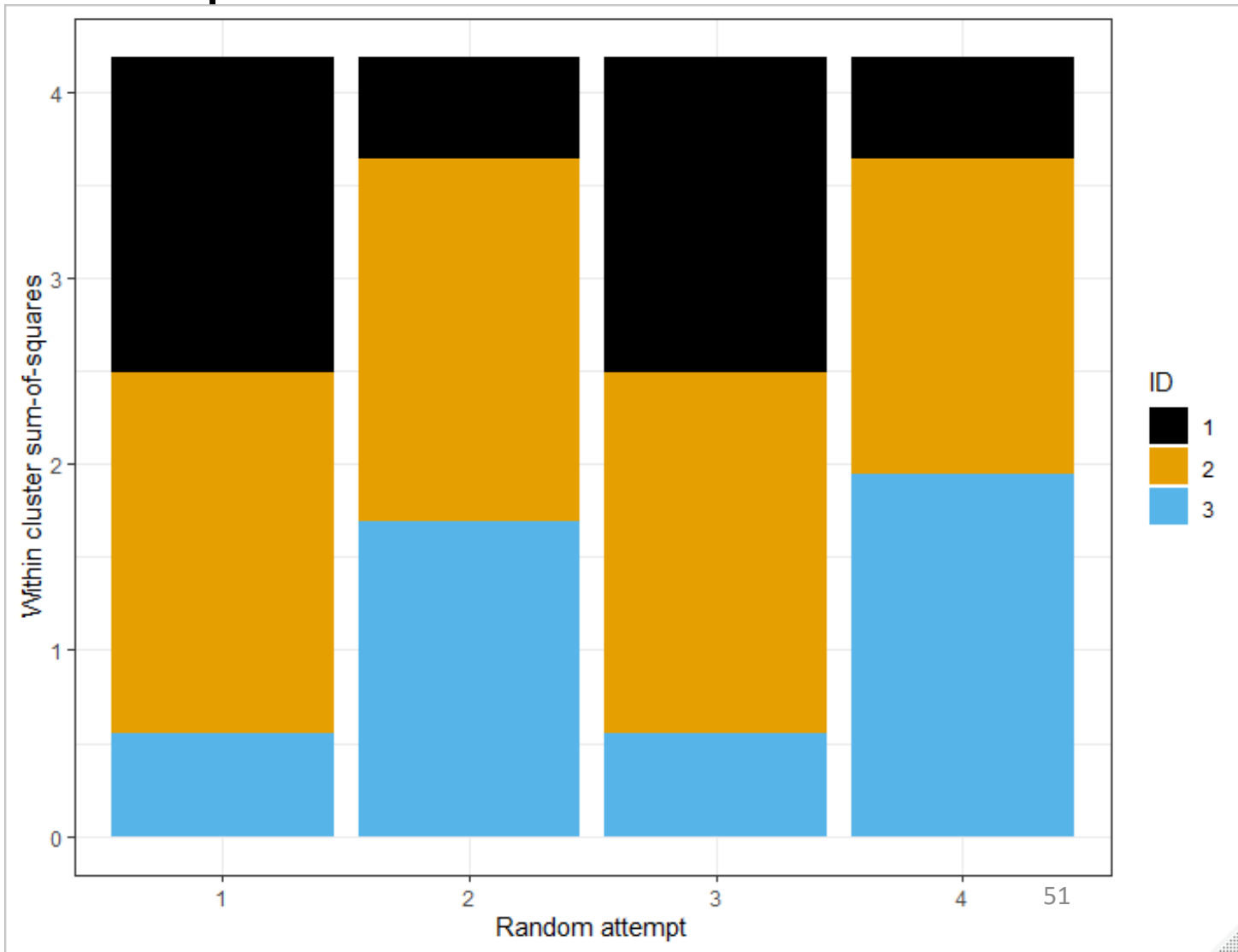
Convergence corresponds to cluster assignments not changing, and total within sum of squares remaining constant across iterations!

Are any of the 4 random attempts better than the others, for this example?

NO!!

The (converged) total within sum of squares is the same across the 4 random starting points.

May not be the case for a real problem.



For this example, we can assess if the cluster assignments are correct

- We know the specific Gaussians each observation was generated from.

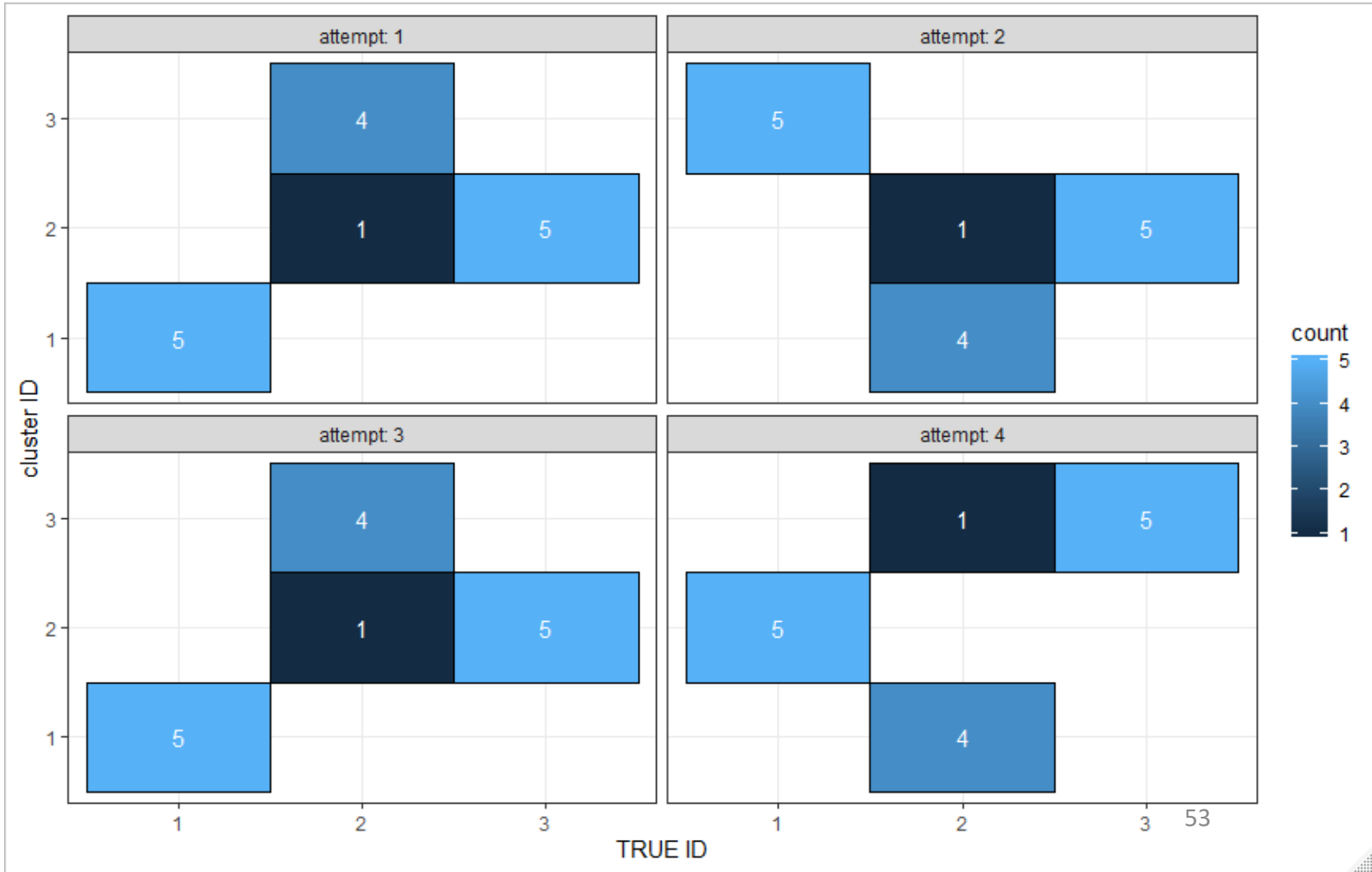
- Comparing cluster results to some available grouping/class variable is a useful way to assess if the clustering algorithm is “working correctly”.

Combinations between the cluster assignments and the TRUE Gaussian group

All 5 observations associated with Gaussians 1 and 3 are correctly grouped together.

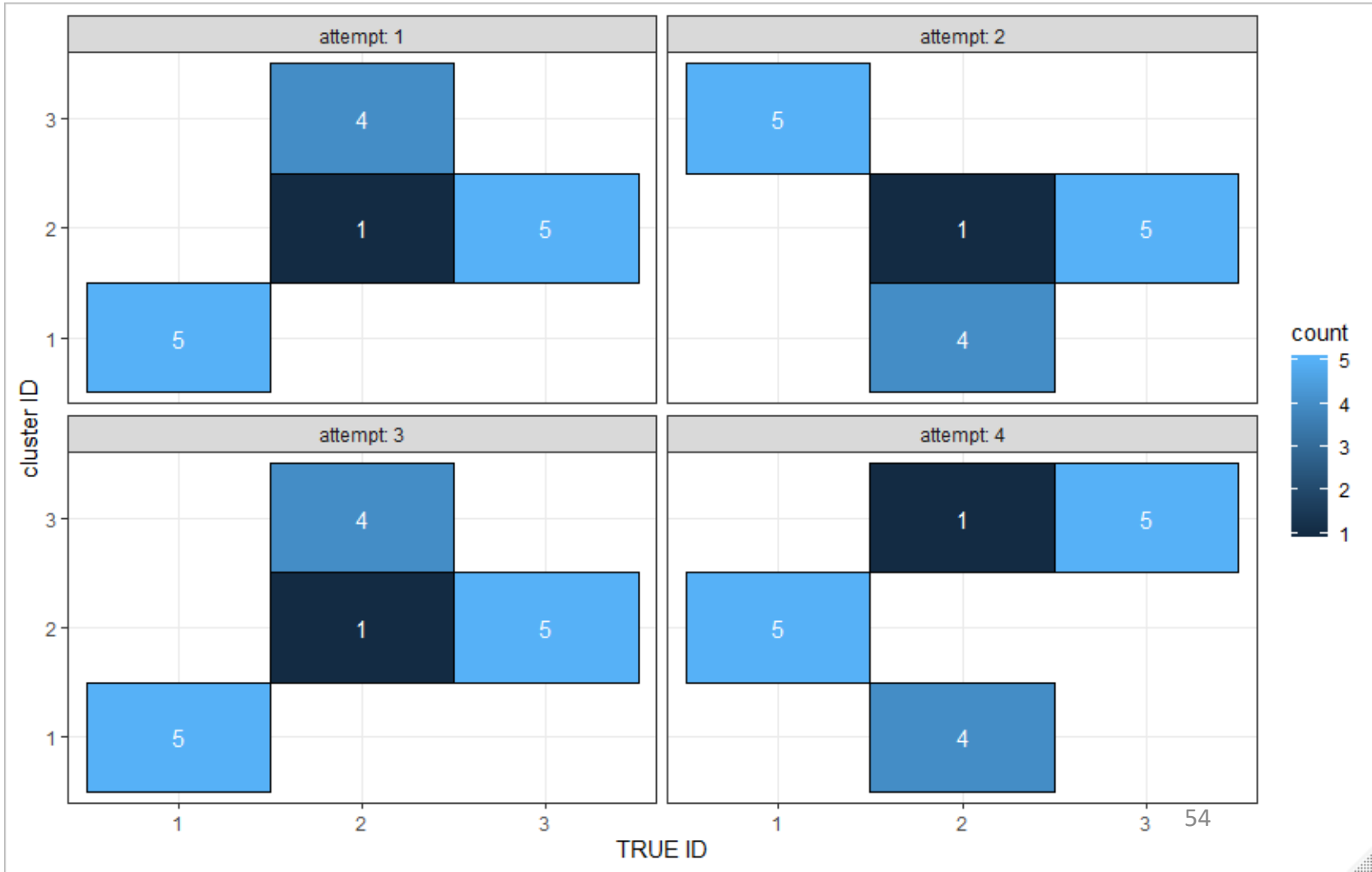
Only 1 observation associated with Gaussian 2 is incorrectly assigned.

That incorrect assignment is attributed to Gaussian 3.



What else does the heat map reveal about the cluster assignments?

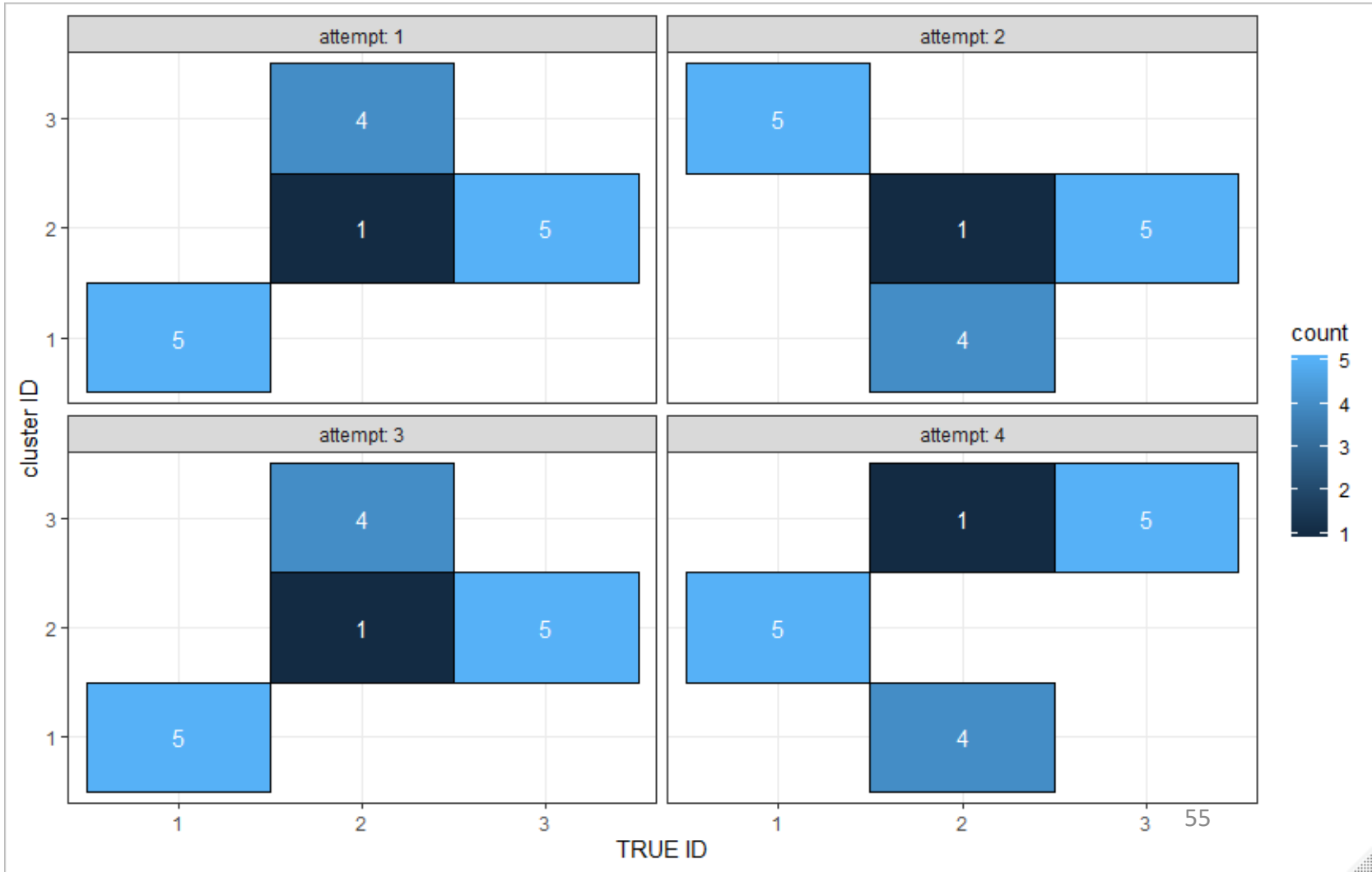
Look at the assignments associated with Gaussians 1 and 3…



Assigned cluster labels are not unique and do not matter!

What matters is that the observations are consistently grouped together across attempts.

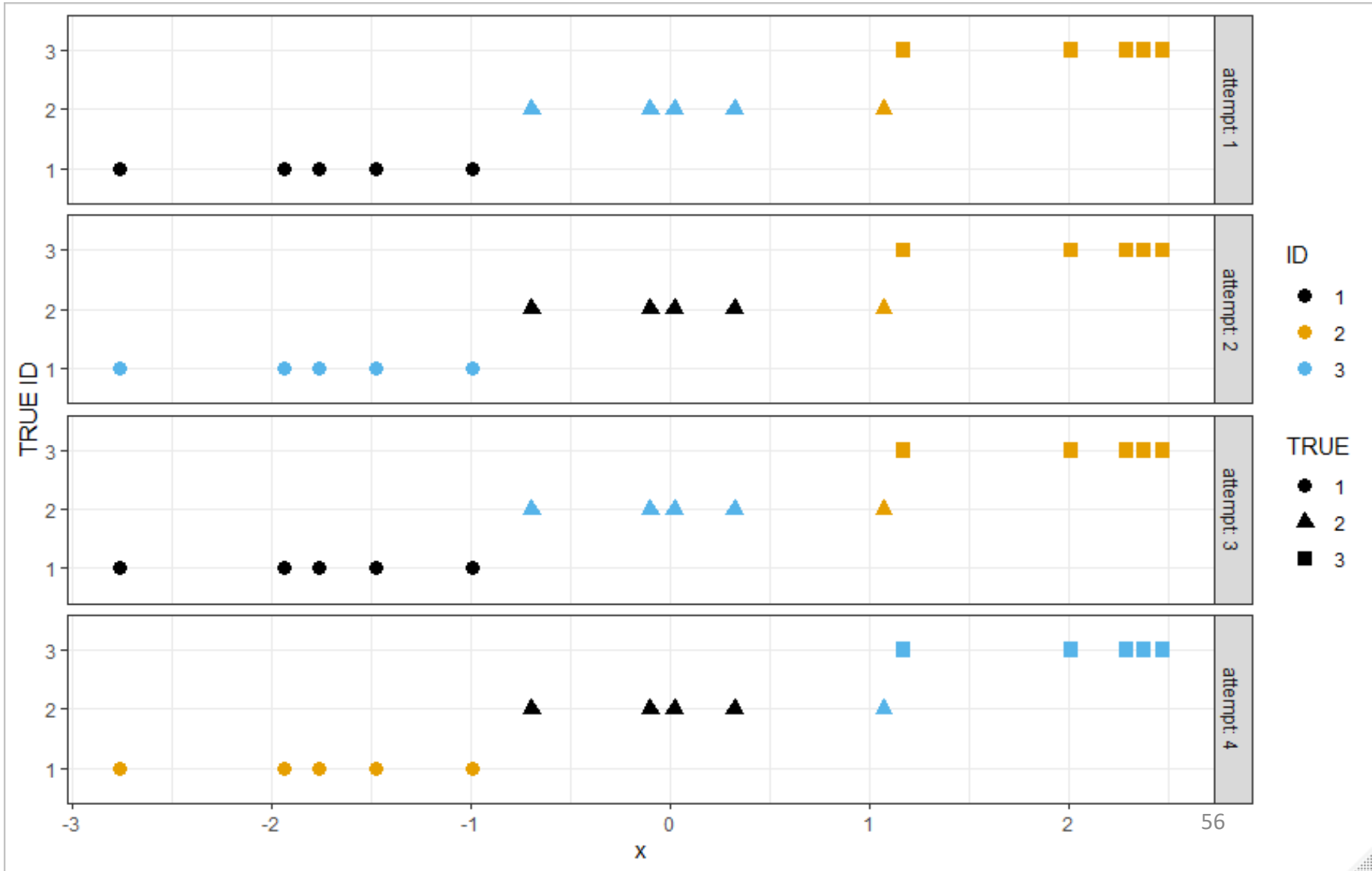
Not if those observations are labeled group 1 or 2.



Which observation was incorrectly assigned?

Vertical axis and marker shape denote the TRUE Gaussian which generated each observation.

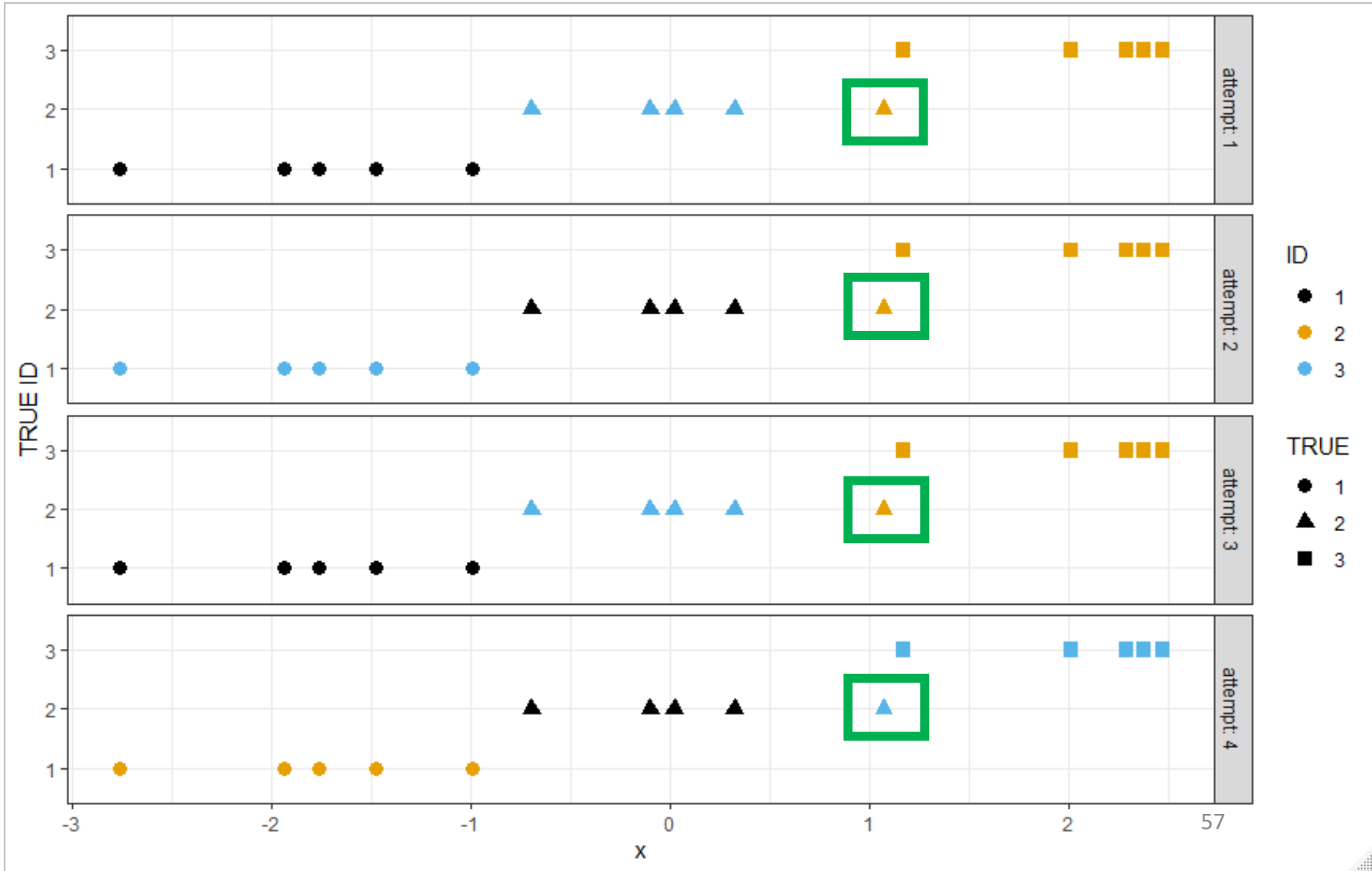
Marker color denotes which cluster KMeans assigned the observation to.



Which observation was incorrectly assigned?

The incorrectly assigned observation is closer to the observations from Gaussian 3 than the other observations from Gaussian 2.

KMeans is judging similarity strictly by Euclidean distance, there's no way for it to "know" that observation would have come from Gaussian 2.

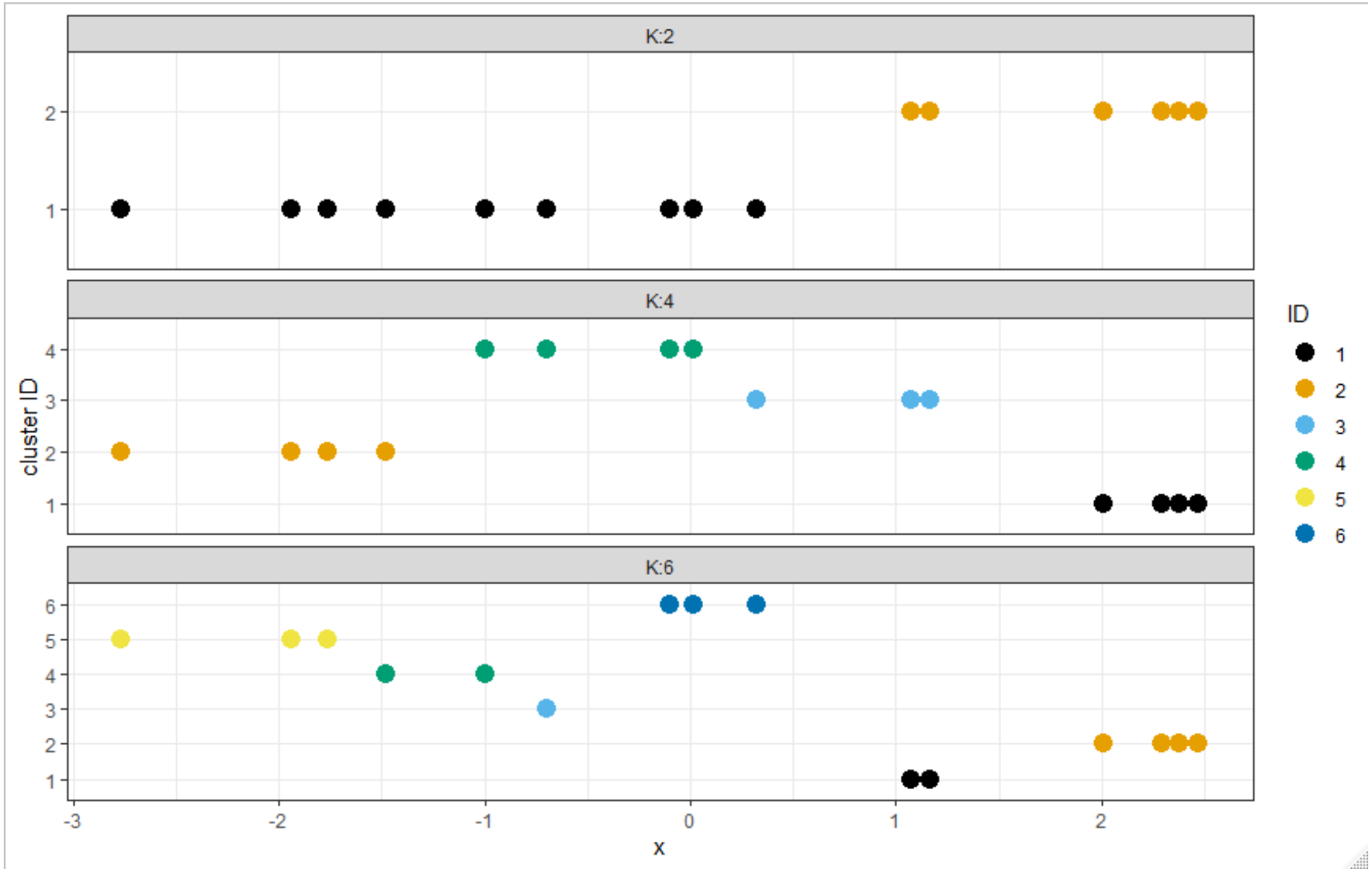


For this problem, we knew to try 3 clusters…

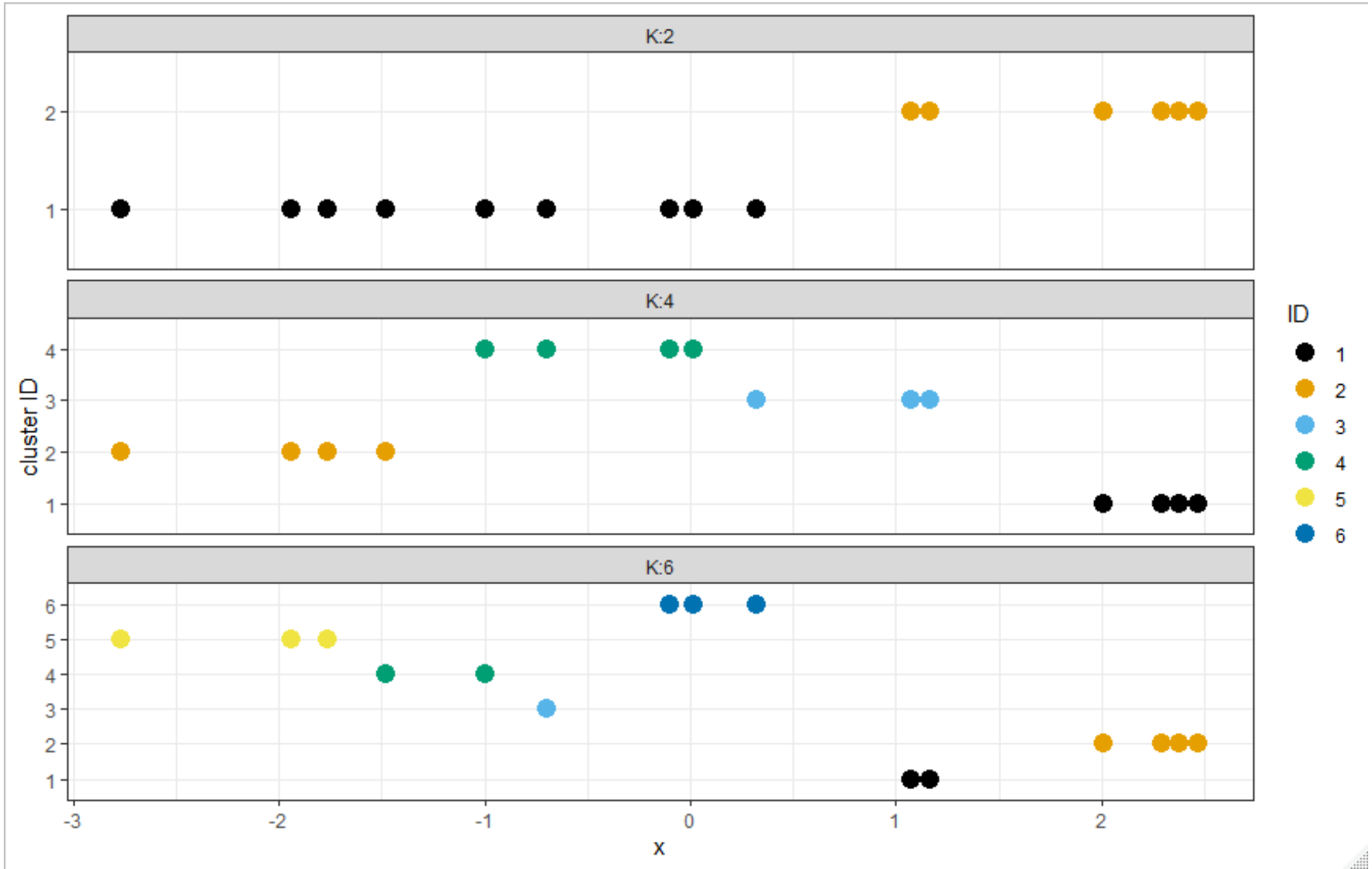
- In a real problem though, we may not have a particular value of K in mind…

- Can we identify the “best” number of clusters?

Look at the cluster results for several other values of K



What do you think happens to the within cluster sum of squares as K continues to increase?

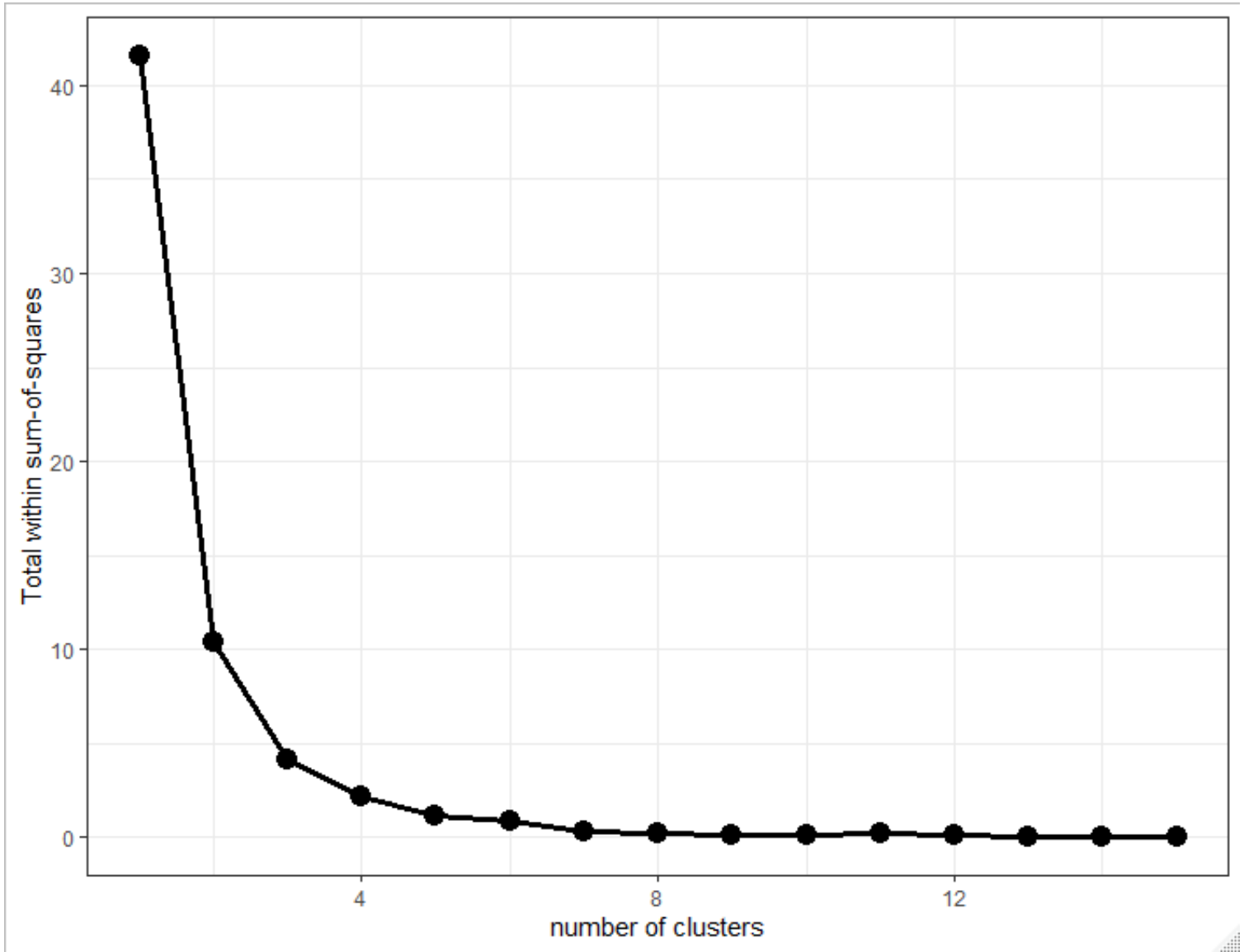


In the limit that each observation is its own cluster…

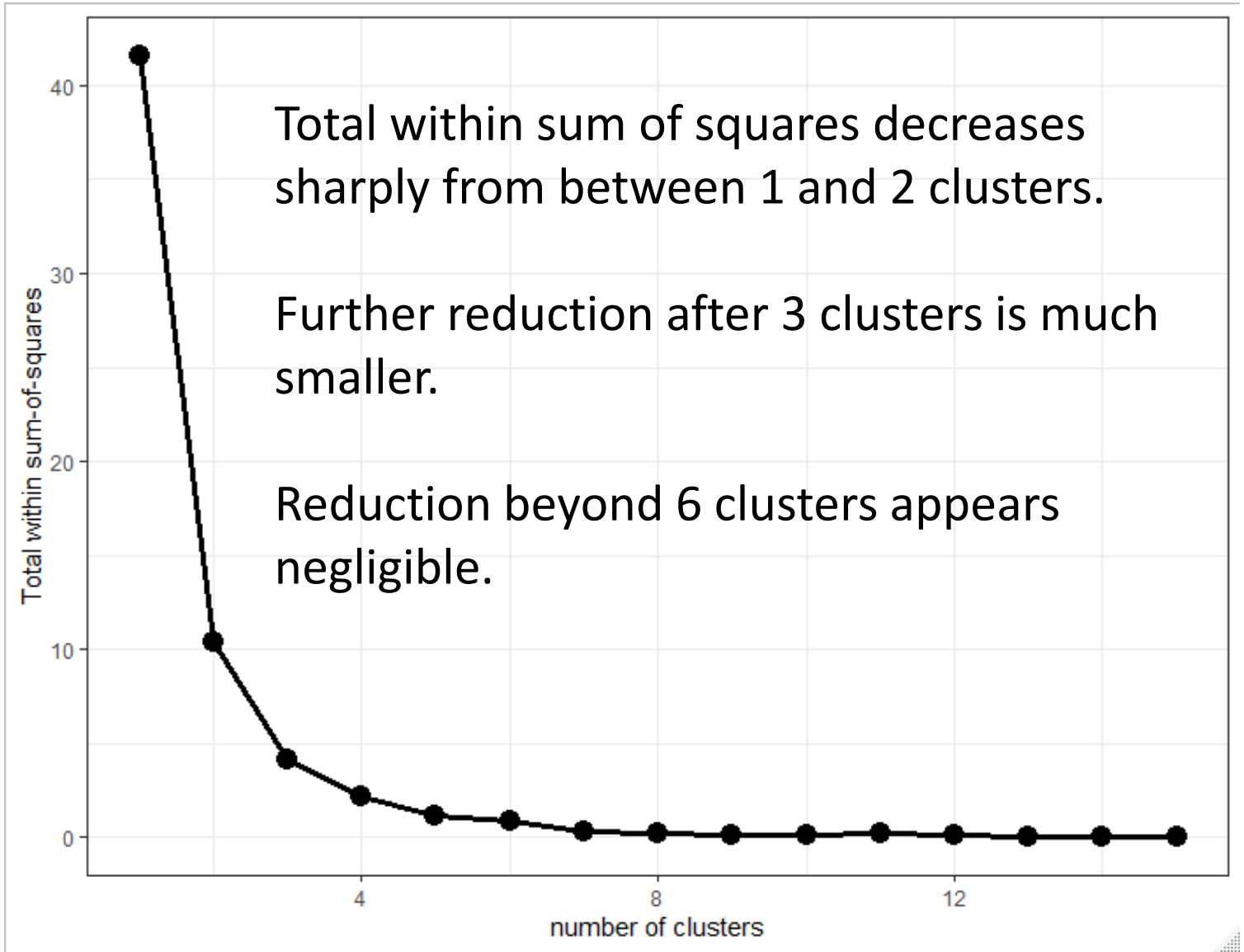
- The within cluster sum of squares is ZERO!

- Thus, we don’t want to arbitrarily drive down the within cluster sum of squares to zero…we need some rules to define “good enough”…

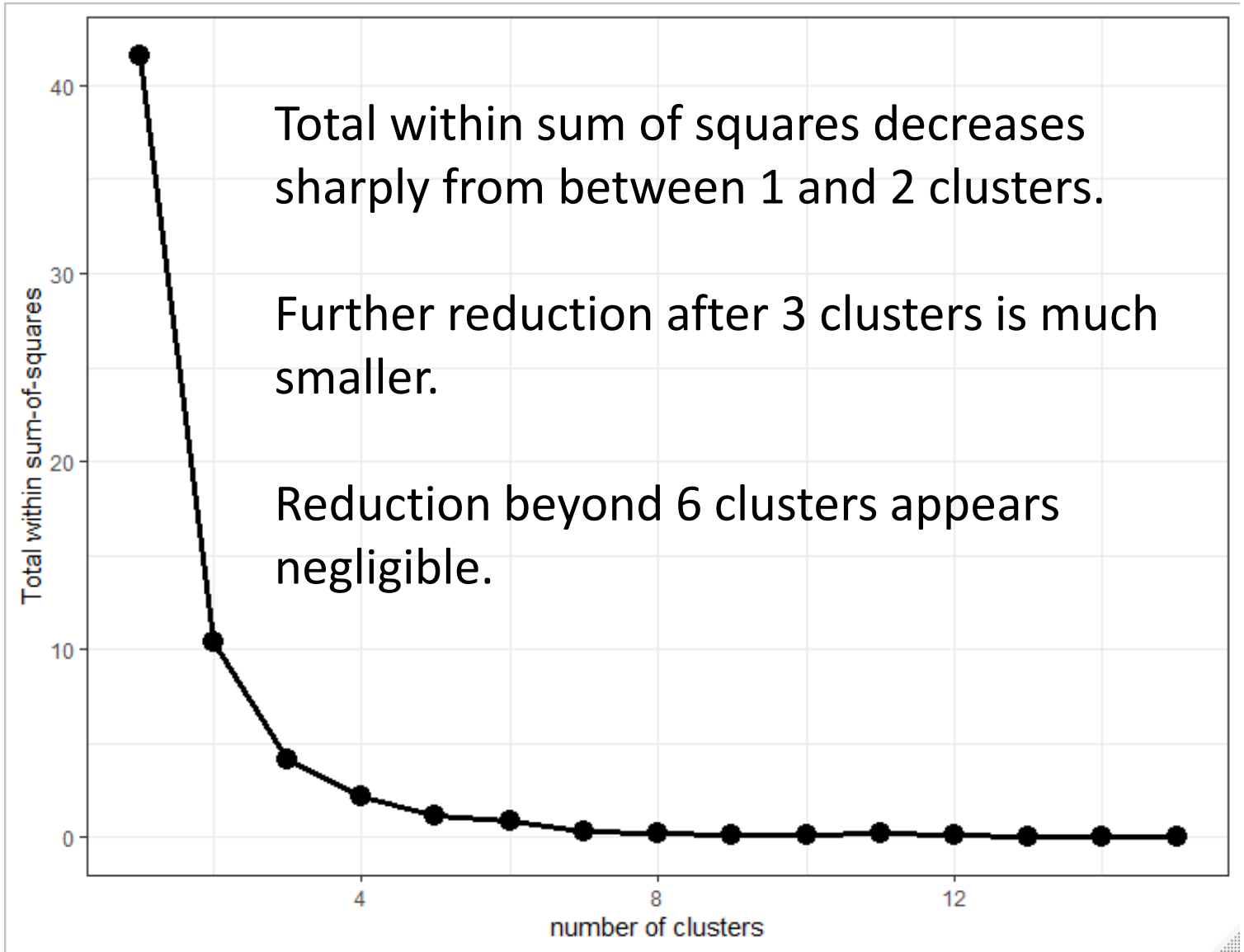
Visual heuristic – look for a knee bend or point of **diminishing returns**



Visual heuristic – look for a knee bend or point of **diminishing returns**



Visual heuristic – Downside…this is rather subjective!



Quantitative metrics for selecting optimal numbers of clusters

- **Hartigan's rule:** compares the ratio of total within sum of squares of k clusters to that of $k + 1$ clusters. If that ratio is greater than a threshold, try $k + 1$ clusters.

- **Gap statistic:** Uses bootstrapping to compare cluster performance to a reference null distribution (uniform). Find the maximum number of clusters that out performs the reference distribution.

- **Silhouette method:** Measure quality of clustering based on how well a point lies within a cluster. Find the maximum number of clusters with the highest average silhouette coefficient.

You can focus on the KNEE BEND visual heuristic for this course.

These additional quantitative metrics are provided show that quantitative metrics do EXIST!

Our example was in 1D…but the concepts hold for higher dimensions!

- Great website to visualizing kmeans clustering in two dimensions:

- https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

- This website initializes the clusters by initializing the centroids, instead of initial cluster assignment. The two ways are basically the same!

Major aspects of KMeans that you MUST remember

- You MUST specify the number of clusters BEFORE running the algorithm!
- Algorithm ITERATES from a starting guess. You must make sure ENOUGH iterations are used.
- We do NOT know the best possible initial guess. We RANDOMLY create the initial guess!
  - Need to try MANY different RANDOM initial guesses!
- Decide the BEST or OPTIMAL number of clusters by calculating the TOTAL WITHIN SUM OF SQUARES for MANY possible number of clusters.
  - Look for a KNEE BEND in the plot to identify the appropriate number of clusters!