

# BIOL 680: Advanced Statistics for Biological Sciences

Guest Lecture: Machine Learning (K-Means)

---

Hammed A. Akande

Quantitative and Community Ecology Lab, Concordia University

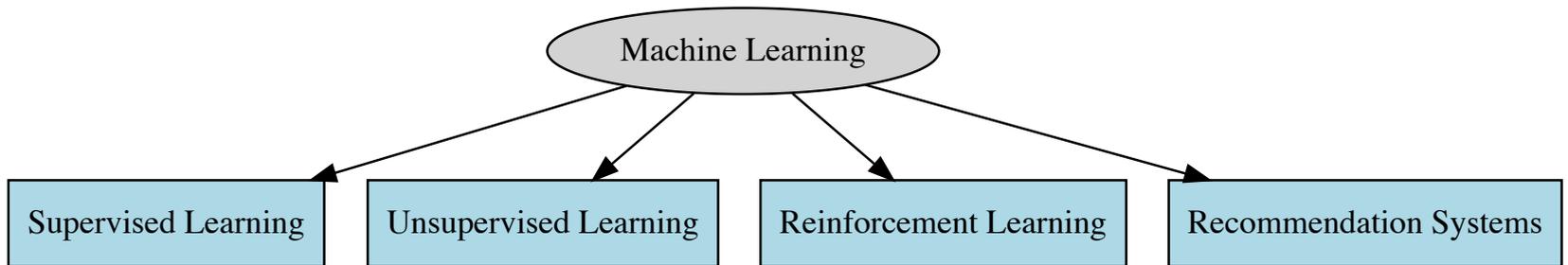
2025-03-18

# K-Means Clustering

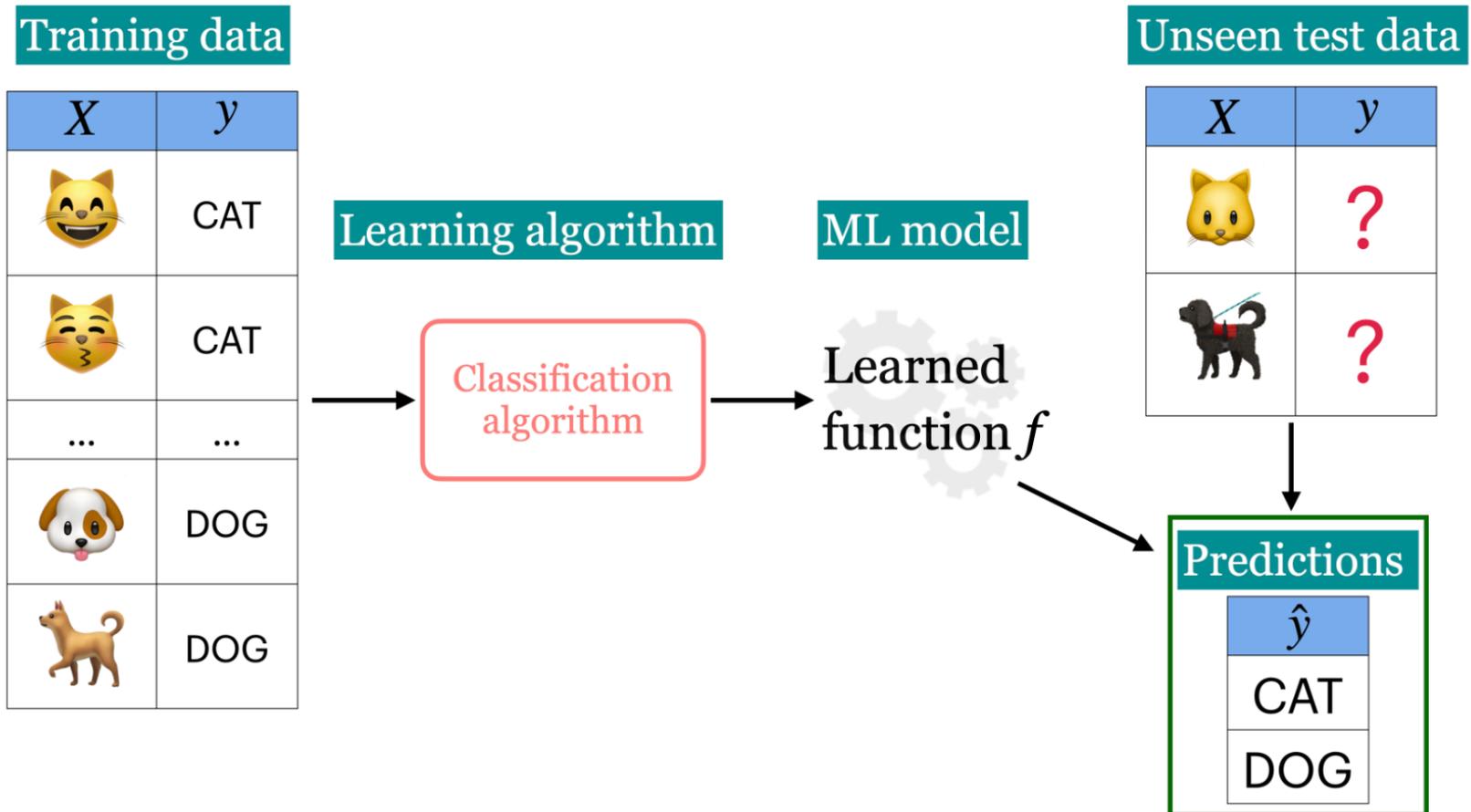
# Goals of the lecture

- **Understand the principles of clustering and K-Means.**
- **Learn how to implement K-Means.**
- **Interpret and evaluate K-Means clustering results.**
- **Explore practical applications of K-Means and best practices.**

# Types of Machine Learning



# Supervised Learning



# Unsupervised Learning

Training data



Learning algorithm

Clustering algorithm

ML model



CATS



DOGS



# Unsupervised Learning

Can we make sense of data without a target variable?

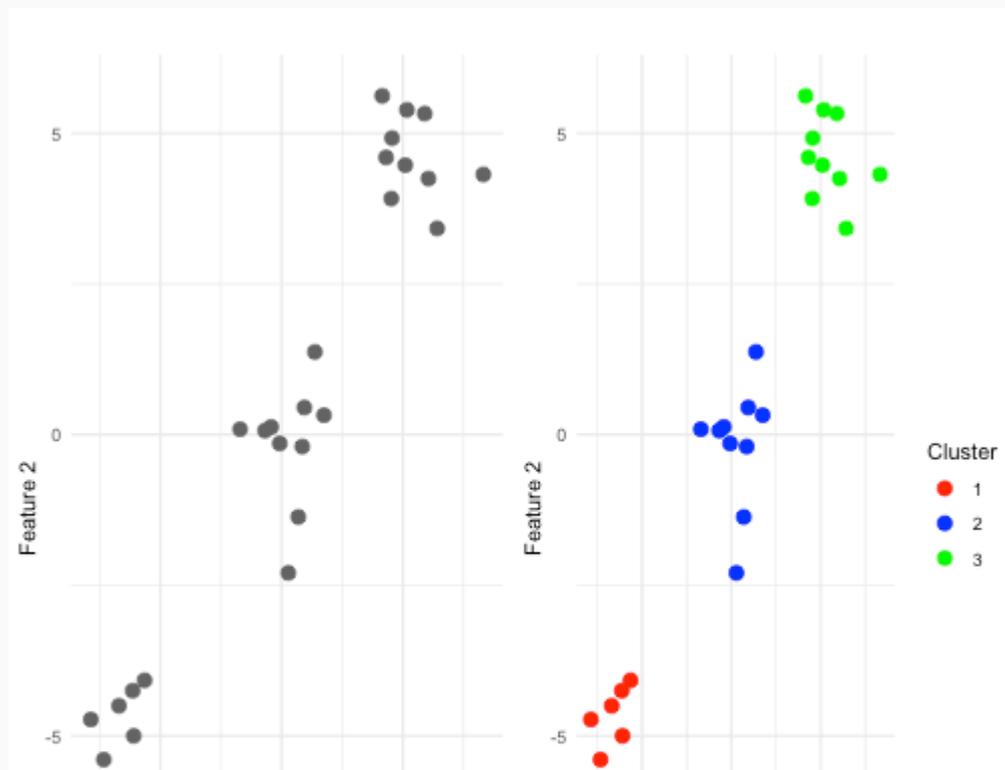
- **Yes**, at least to some extent. Although our focus would shift to finding the underlying structure of the input data instead of function between input and output like in supervised learning.
- Examples include:
  - Clustering (e.g., K-Means)
  - Dimensionality Reduction (e.g., PCA)

# Clustering

# Clustering

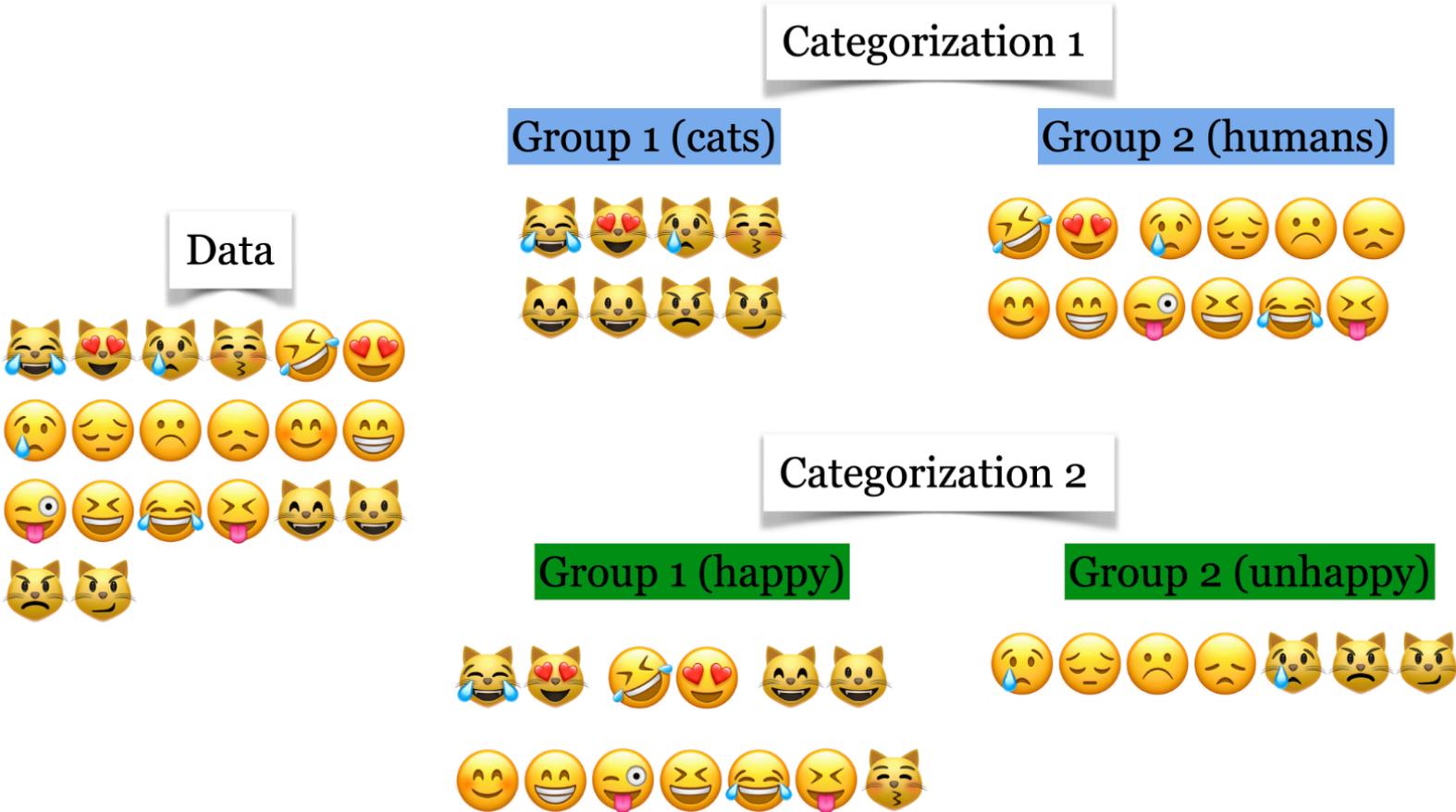
Partitioning data into groups called clusters based on their similarities.

- The goal is to discover underlying groups/patterns in a given dataset such that:
  - data in the same group are as similar as possible.
  - data in different groups are as different as possible.



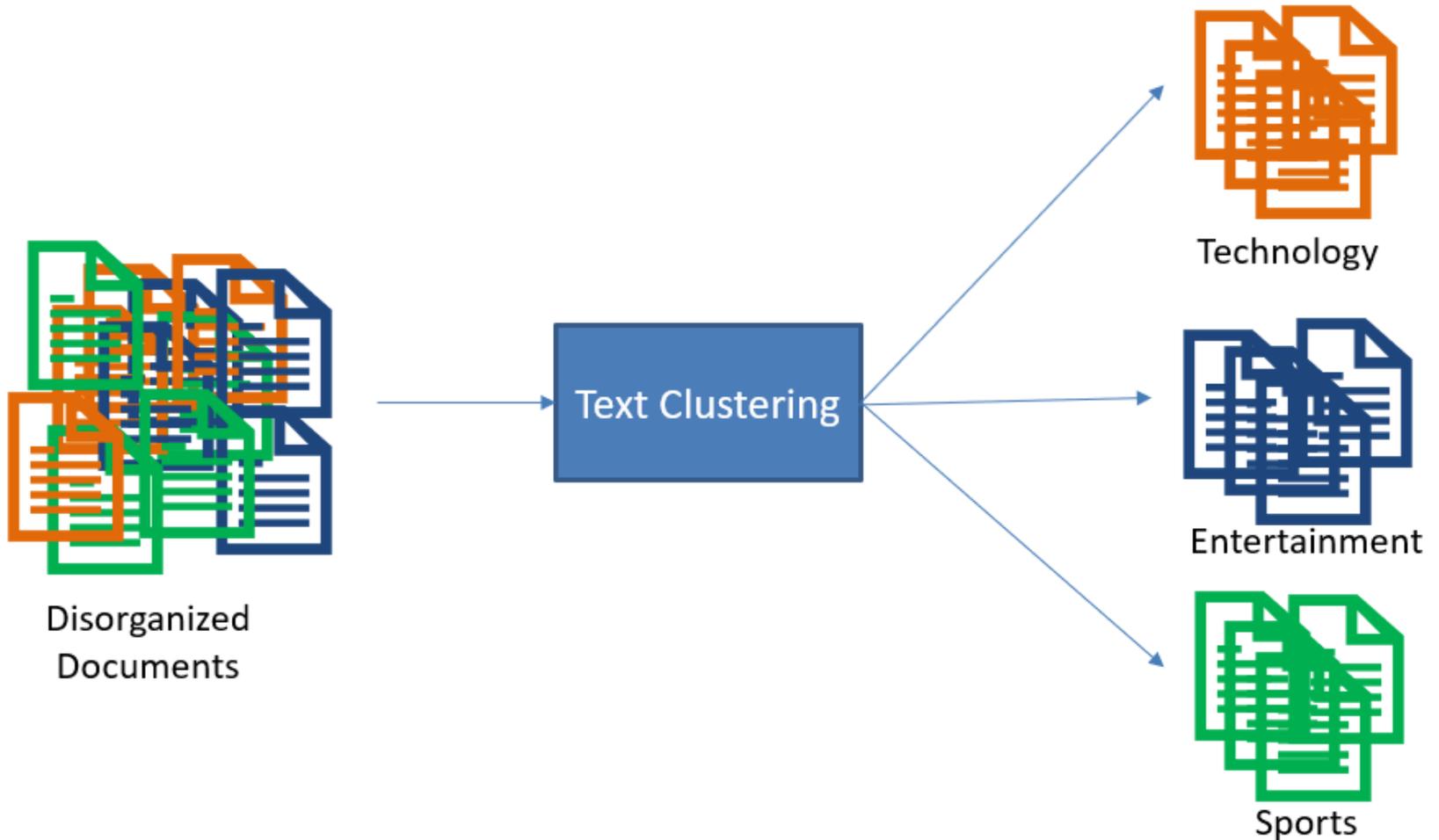
# Clustering

- What is the "correct" grouping here?



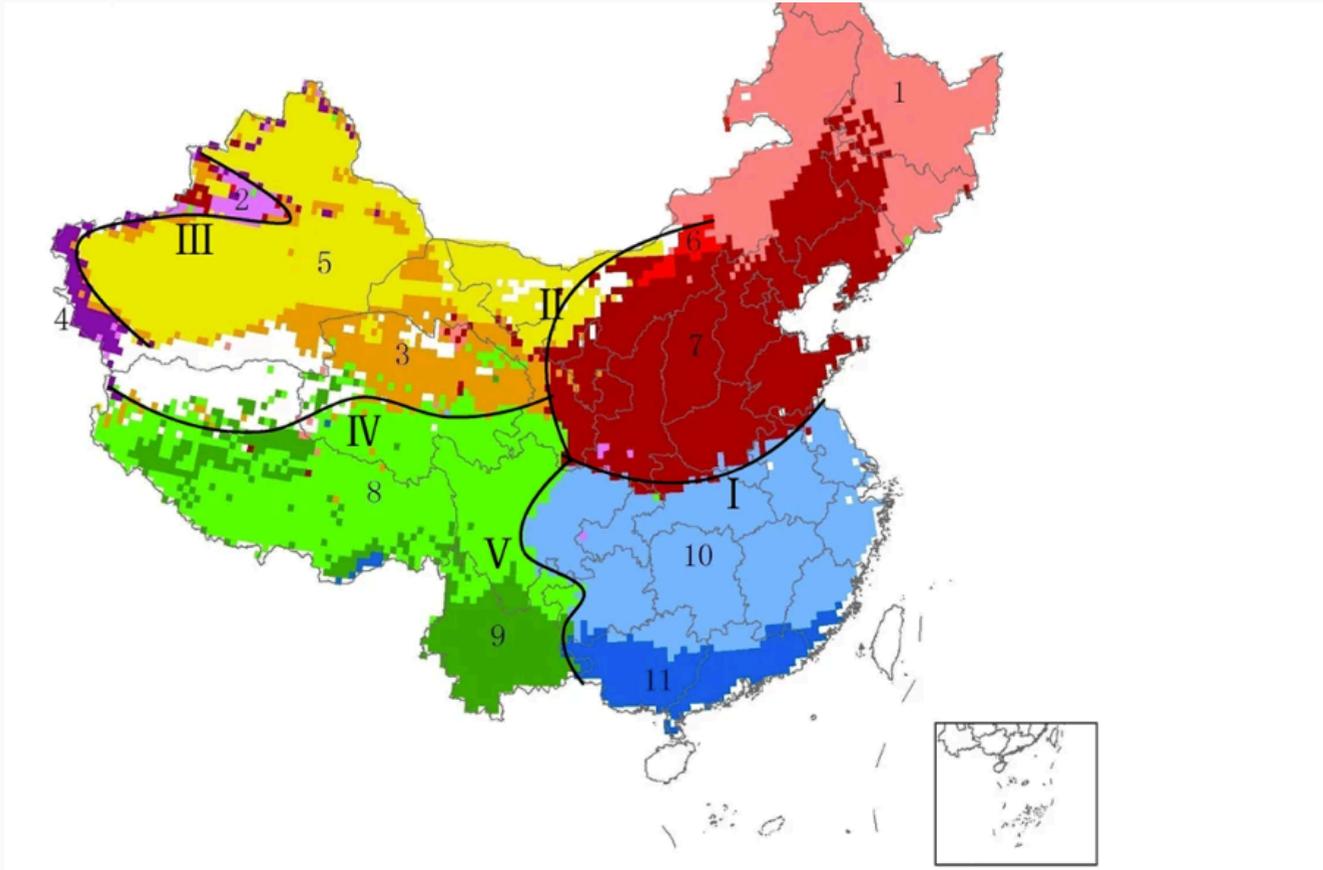
# Use Cases: Document Clustering

Group different topics from different articles or new sources.



# Use Cases: Species Distribution and Habitat Classification

We can cluster ecoregions based on environmental variables (temperature, precipitation, etc) to identify distinct habitat types.



# Other Common Applications

- **Genetic and Genomic Clustering:** cluster genes with similar expression patterns in transcriptomics studies.
- **Social Network Analysis**
- **Customer segmentation**

# K-Means clustering

# K-Means clustering

K-Means is among the most widely used algorithms for clustering data

- **Input**

- $X$  -> set of data points
- $k$  -> number of clusters

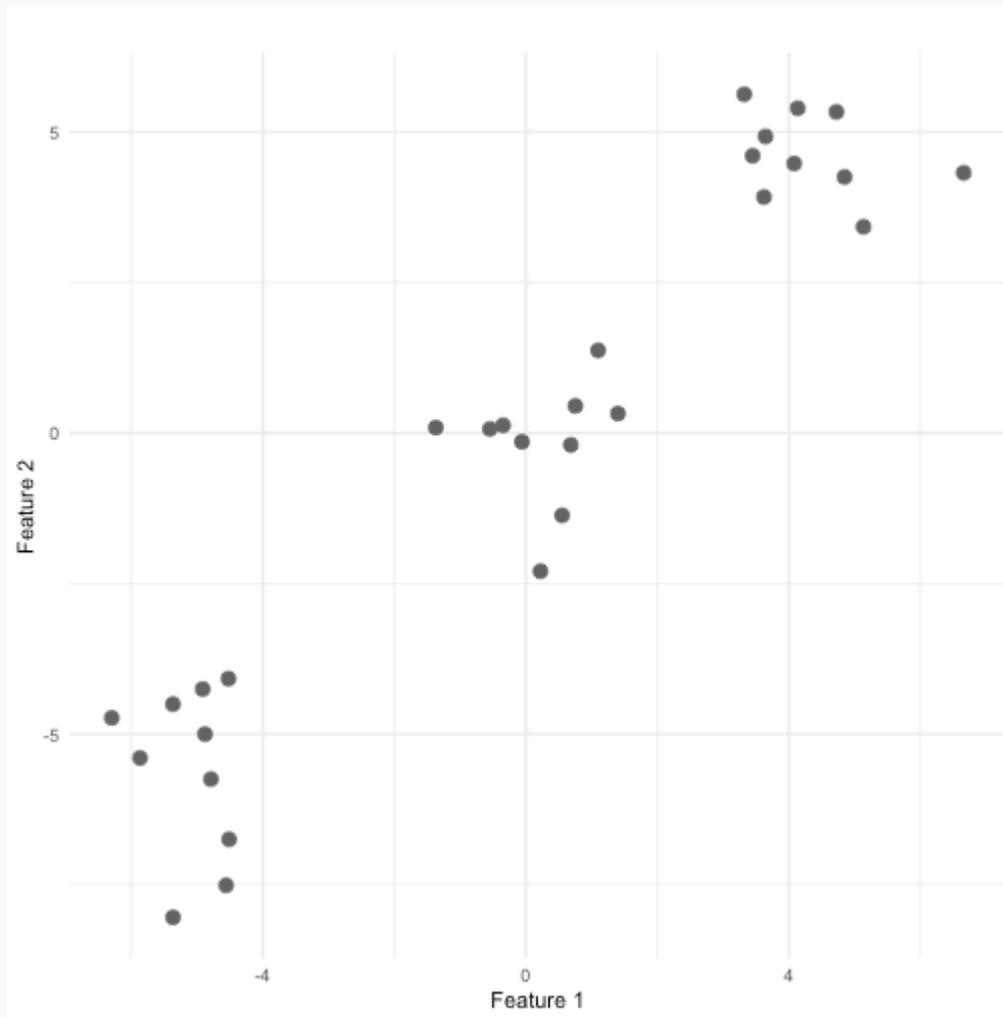
- **Output**

- $k$  clusters of the data points

# KMeans

```
##           X1           X2
## 1 -4.78603750 -5.74576905
## 2 -4.52034187 -4.07844964
## 3 -4.91217130 -4.24994565
## 4 -4.55614149 -7.50855402
## 5 -5.36283792 -8.04093410
## 6 -4.87732597 -4.99973420
## 7 -5.86384519 -5.39401899
## 8 -4.51037573 -6.74502766
## 9 -5.36411691 -4.50136855
## 10 -6.29424201 -4.72904621
## 11  1.09892152  1.37305395
## 12  0.75251346  0.45025656
## 13 -0.05941669 -0.14629386
## 14 -0.34456879  0.12809724
## 15  0.22266830 -2.29472095
## 16  0.55178634 -1.36656892
## 17  0.68364282 -0.19747955
## 18 -0.54587940  0.06808578
## 19 -1.36743616  0.09050341
## 20  1.40005184  0.32275997
```

# KMeans



# KMeans

```
# K-means clustering
kmeans_result ← kmeans(X, centers = 3, nstart = 10)

kmeans_result$cluster
```

```
## [1] 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3
```

The output of KMeans is `n_clusters` (groups) of the data points

Now, we can retrieve the cluster assignment for each observation and add them to the data.

```
## # A tibble: 9 × 3
##   feat1 feat2 `cluster labels`
##   <dbl> <dbl>         <int>
## 1 -0.345  0.128             1
## 2  0.684 -0.197             1
## 3  1.10   1.37              1
## 4 -4.79  -5.75             2
## 5 -4.88  -5.00             2
## 6 -4.91  -4.25             2
## 7  4.08   4.47             3
## 8  4.13   5.39             3
## 9  4.84   4.25             3
```

# KMeans

In KMeans, each cluster is usually denoted by its center/centroid



# KMeans Prediction

```
# predict cluster assignments for new data points
predict_kmeans ← function(model, newdata) {
  centers ← model$centers
  distances ← as.matrix(dist(rbind(centers, newdata))[-(1:nrow(centers)), 1:nrow(centers)])
  new_labels ← max.col(-distances)
}

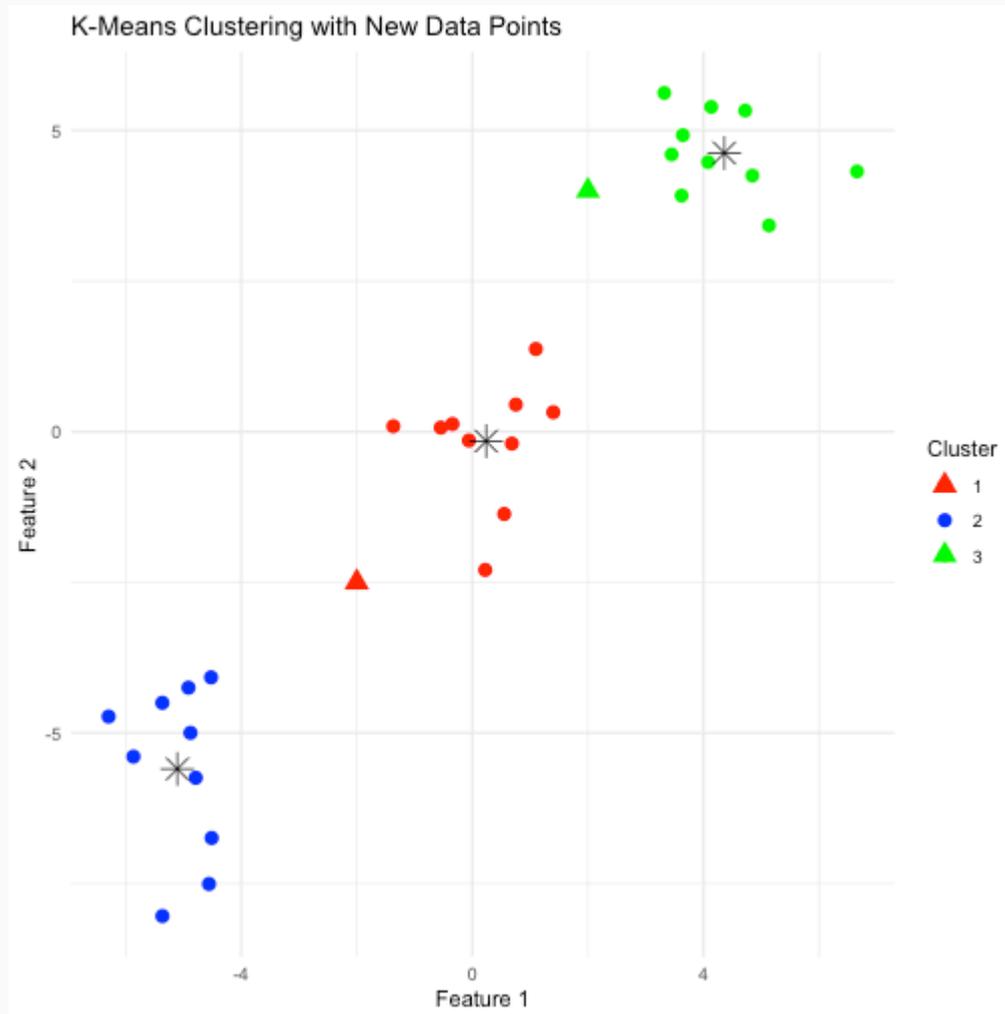
# New data points
new_examples ← matrix(c(-2, -2.5, 2, 4), ncol = 2, byrow = TRUE)

# Predict cluster assignments for new data points
new_labels ← predict_kmeans(kmeans_result, new_examples)
new_labels

## [1] 1 3
```

# KMeans Prediction

Now, we can visualize the clusters, centers and new data points



# K-Means algorithm

In K-Means clustering, the objective is to represent each cluster by its centroid and assign each data point to a cluster.

But this is a bit complex in reality:

- **Assigning Data Points:** If the cluster centers were known, each data point could be assigned to the nearest center.
- **Determining Cluster Centers:** Conversely, if the cluster assignments were known, the centers could be computed as the mean of the assigned points.

The problem, however, is we neither know the cluster centers nor the assignments(a priori).

So, how do we resolve that?

***Iterative approach***

# KMeans algorithm

Recall,

**Input:** our data points ( $X$ ) and the number of clusters ( $K$ )

**Initialization:** Randomly select initial cluster centers.

**Iterative process:**

- **Assignment Step:** Assign each data point to the nearest cluster center.
- **Update Step:** Recalculate the cluster centers as the mean of the assigned points.
- **Convergence Check:** Repeat the above steps until the assignments no longer change or maximum iterations reached.

# Example

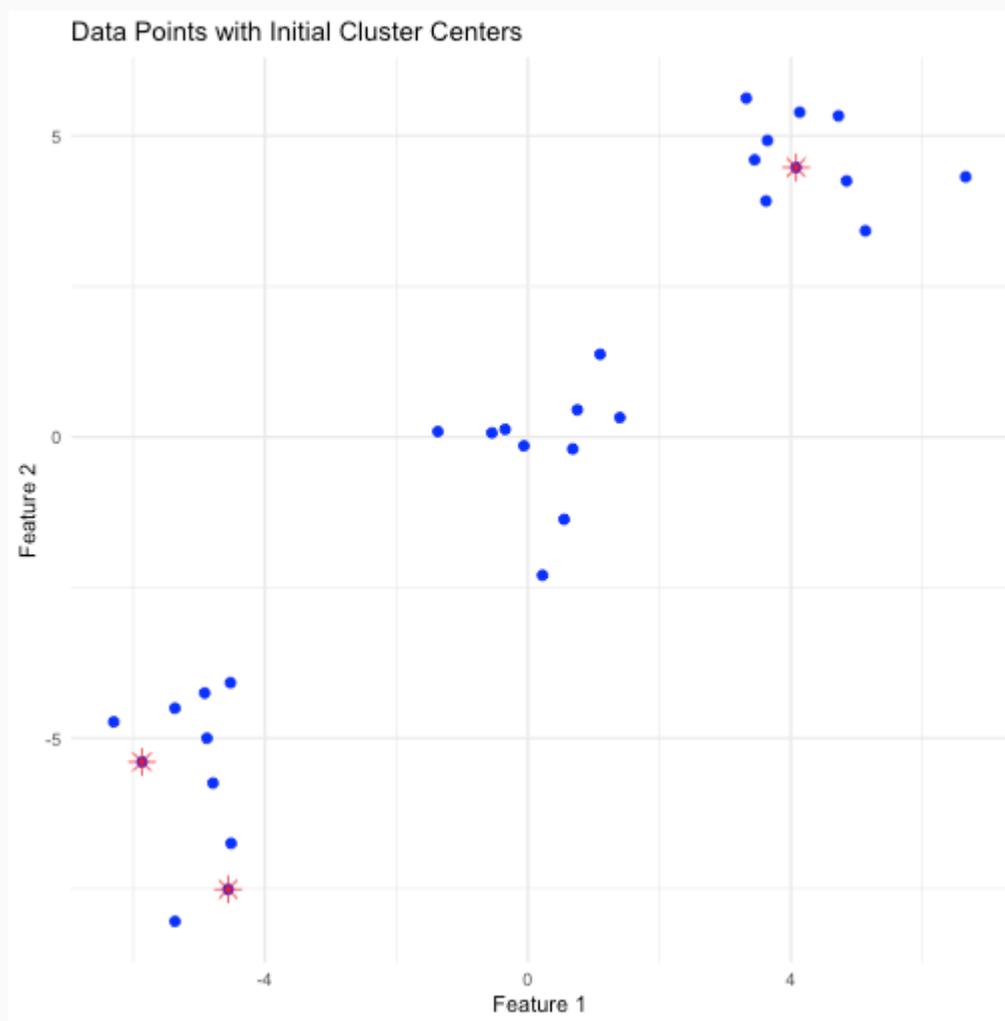
So, Let's execute K-Means algorithm on our simulated data.

```
## Number of examples: 30

##           [,1]      [,2]
## [1,] -4.786037 -5.745769
## [2,] -4.520342 -4.078450
## [3,] -4.912171 -4.249946
## [4,] -4.556141 -7.508554
## [5,] -5.362838 -8.040934
## [6,] -4.877326 -4.999734
## [7,] -5.863845 -5.394019
## [8,] -4.510376 -6.745028
## [9,] -5.364117 -4.501369
## [10,] -6.294242 -4.729046
```

Recall, that  $k = 3$

# Initialization



# Next: Iterative process

repeat

- **Assignment Step:** Assign each data point to the nearest cluster center.
- **Update Step:** Recalculate the cluster centers as the mean of the assigned points.
- **Convergence Check:** repeat the steps until the assignments no longer change

# How do we find closest centers?

- First step is to assign examples to the closest center.
- We can consider distance of data to all centers and assign that to the closest center.

Let's see this example.

## Closest Centers Example

# Finding Closest Centers

```
# calculate distances and update cluster assignments
update_Z ← function(X, centers) {
  X ← as.matrix(X)
  centers ← as.matrix(centers)

  # Number of data points and centers
  n_points ← nrow(X)
  n_centers ← nrow(centers)

  # Initialize a matrix to store distances
  dist_matrix ← matrix(0, nrow = n_points, ncol = n_centers)

  # Calculate Euclidean distances
  for (i in 1:n_points) {
    for (j in 1:n_centers) {
      dist_matrix[i, j] ← sqrt(sum((X[i, ] - centers[j, ])^2))
    }
  }

  # nearest center for each point
  cluster_assignments ← apply(dist_matrix, 1, which.min)

  return(list(distances = dist_matrix, assignments = cluster_assignments))
}
```

# Update centers

- Now that we have new cluster assignments, we need to update cluster centers.
- New cluster centers are means of data points in each cluster.

```
update_centers ← function(X, Z, old_centers, k) {  
  # Initialize new centers  
  new_centers ← old_centers  
  
  # Iterate over each cluster  
  for (kk in 1:k) {  
    # Identify the indices of data points assigned to cluster kk  
    cluster_indices ← which(Z == kk)  
  
    # Calculate the mean of these data points to update the cluster center  
    if (length(cluster_indices) > 0) {  
      new_centers[kk, ] ← colMeans(X[cluster_indices, ], drop = FALSE)  
    } else {  
      # If a cluster has no points assigned, retain the old center  
      new_centers[kk, ] ← old_centers[kk, ]  
    }  
  }  
  
  return(new_centers)  
}
```

# Putting Together

- Initialize
- Iteratively alternate between the following two steps.
  - Assignments update  $z$  -> we assign each data to the closest center
  - Center Update -> we estimate new centers as average of data in a cluster

# Iterations

## Iterations Example

# KMeans: When do we stop?

- When the centroids are no longer changing
- That indicates KMeans convergence, thus we stop!
- KMeans will eventually converge, but doesn't necessarily mean it find the "*right*" clusters. It can converge to local optima - the final clusters depend on the starting positions of the centroids.

# Initialization of K-Means is stochastic

- Because the initial cluster centers are selected randomly
  - This means that if you run the K-Means multiple times with different random initializations, you might get **different clustering results**.
- 

## *Initialization is Important*

- K-Means can converge to local optima — the final clusters depend on the starting positions of the centroids.
- Poor initialization can lead to:
  - Slow convergence.
  - Incorrect clustering.
  - Higher final inertia (sum of squared distances from points to cluster centers will be larger).

How do we decide the value of  $K$ ?

# Hyperparameter Optimization

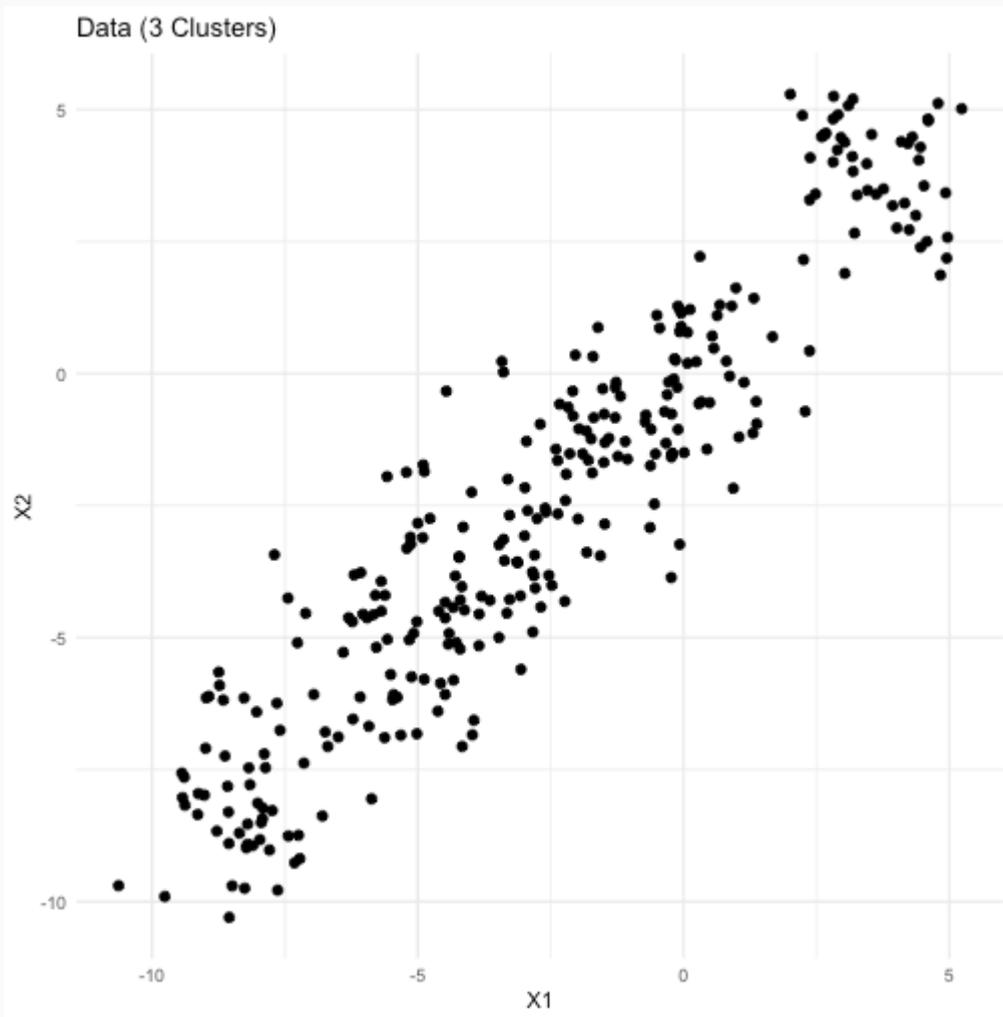
- The number of clusters must be defined before running K-Means.
- For example, in supervised learning, hyperparameter tuning can be done by cross-validation.
- However, In unsupervised learning, since we don't have target values, it's harder to evaluate the algorithm's performance objectively.
- There's no single perfect method for choosing the optimal number of clusters.
- However, there are several strategies that can help estimate a suitable value for K.

# The Elbow method

- This method evaluates the total sum of distances within each cluster, known as inertia
- Inertia represents the sum of squared distances between each data point and its assigned cluster center.
- the intra-cluster distance can be expressed as:

$$\sum_{P_i \in C_1} \text{distance}(P_i, C_1)^2 + \sum_{P_i \in C_2} \text{distance}(P_i, C_2)^2 + \sum_{P_i \in C_3} \text{distance}(P_i, C_3)^2$$

# Inertia



# Inertia

# Inertia

```
##      K  Inertia
## 1    1 9519.7190
## 2    2 3030.2442
## 3    3 1559.8330
## 4    4  828.0066
## 5    5  634.7676
## 6    6  519.9277
## 7    7  467.0376
## 8    8  416.6967
## 9    9  376.1895
## 10 10  339.3888
```

- From the above table, as  $k$  increases, *inertia* decreases

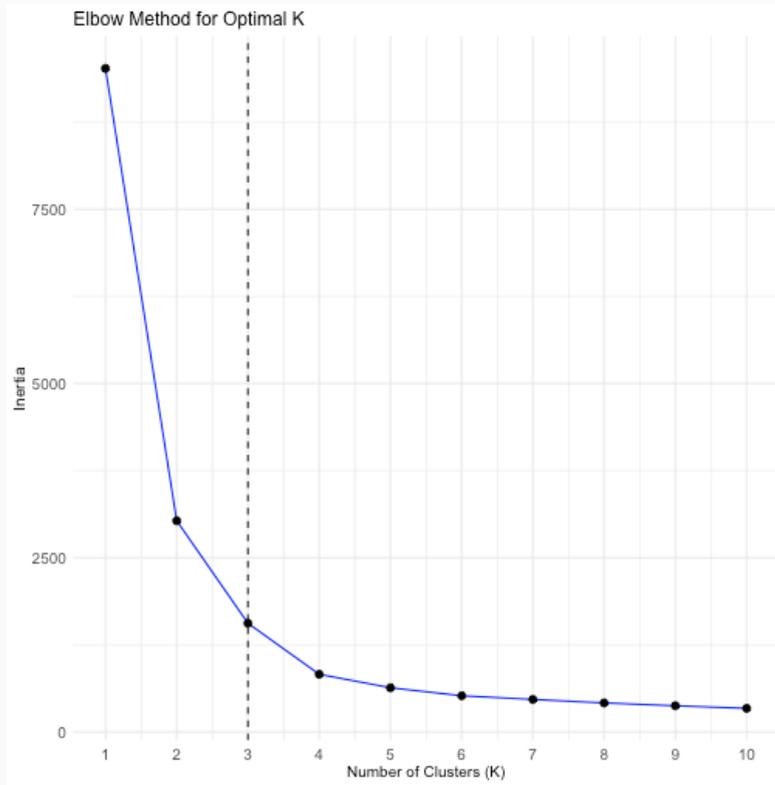
## So, should the inertia be small or very large?

- Actually, the goal is not to just look for a  $k$  that minimizes inertia since it decreases as  $k$  increases.

For example:

- If we have number of  $k =$  number of data points, each data will have its own cluster and then intra-cluster distance will  $= 0$ .
- So, trade-off is necessary here: i.e., we need small  $k$  vs small intra-cluster distances.

# Inertia



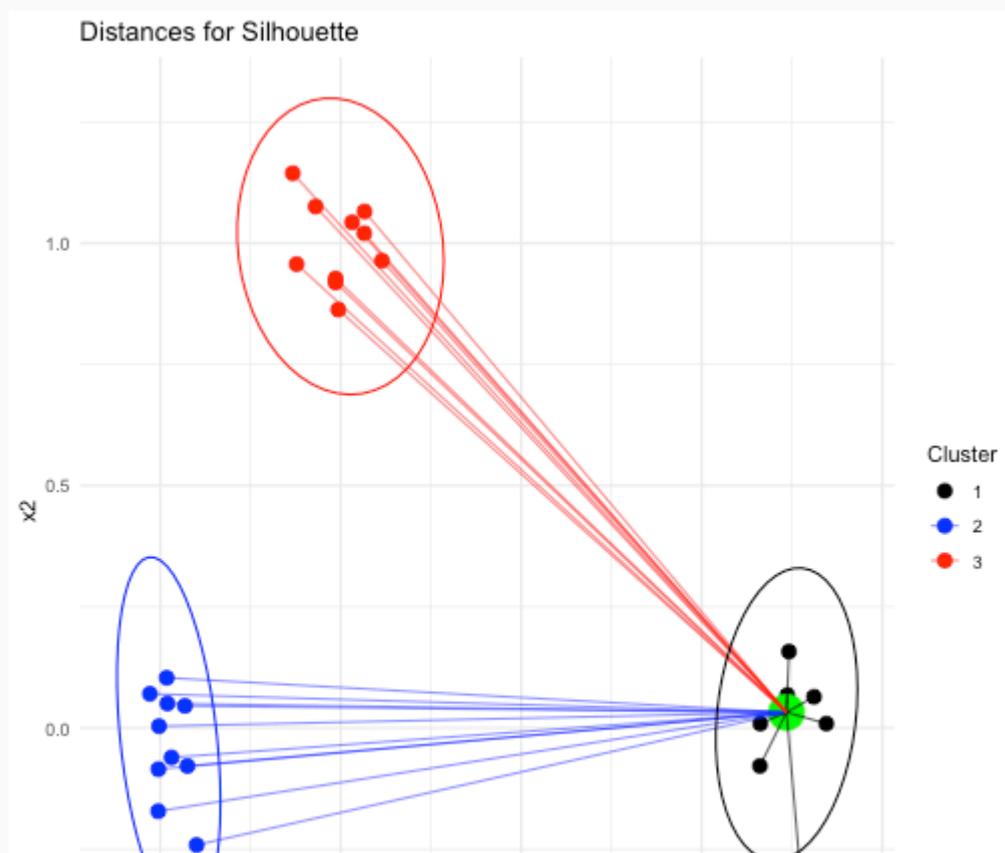
- The plot suggests that three clusters should be okay, as indicated by the inflection point on the curve.
- Although inertia decreases with more than three clusters, the improvement is minimal, so  $K = 3$  is a reasonable choice.
- In our example here, the plot is straightforward to interpret, but it can be challenging in real-world scenarios.

# Silhouette method

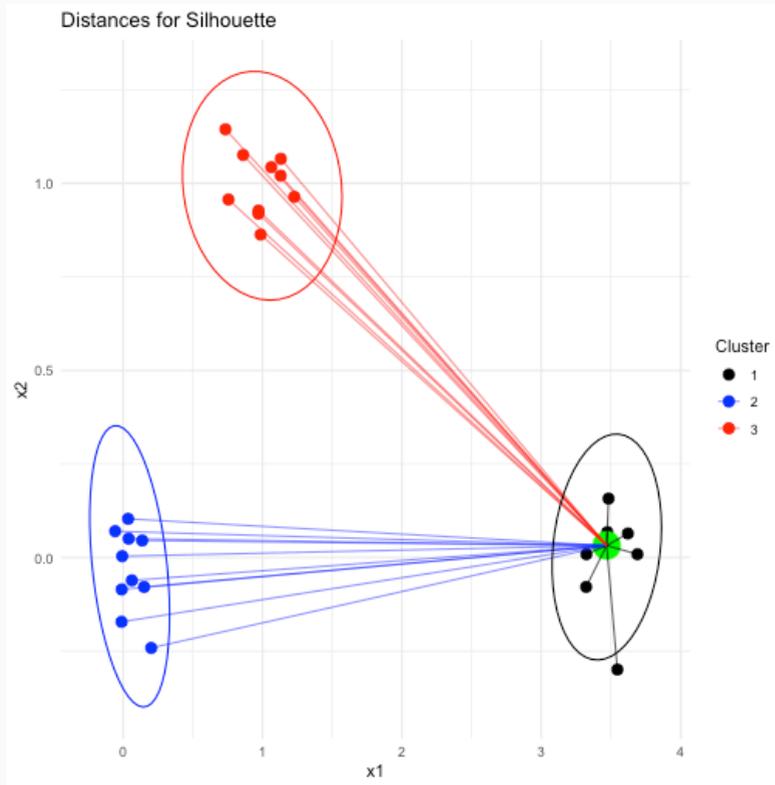
- Does not rely on the concept of cluster centroids.
- Computed using two metrics for each data point:
  - The mean distance to all other points within the same cluster (a).
  - The mean distance to all points in the nearest neighboring cluster (b).

# Mean intra-cluster distance (a)

- For a given sample (e.g., the green point here), calculate the average distance to all other points within the same cluster.
- These distances are depicted by black lines connecting the sample to other points in its cluster.



# Mean nearest-cluster distance (b)



- The average distance from the green point to the blue points is smaller than to the red points, making the blue cluster the **closest**.
- Thus, the mean nearest-cluster distance = average distance between the green point and the blue points.

# Silhouette Score for samples

- The silhouette distance for a sample is the difference between the mean nearest-cluster distance ( $b$ ) and the mean intra-cluster distance ( $a$ ), normalized by the maximum of the two values:

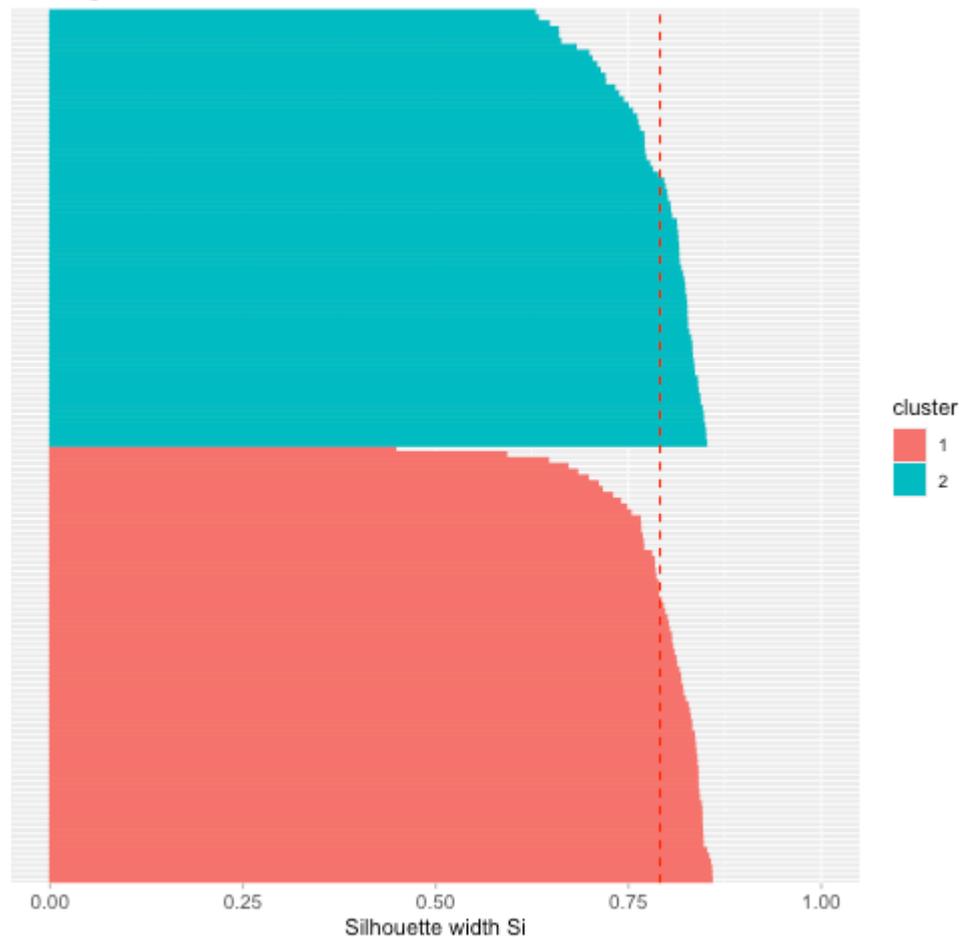
i.e.,

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

- It ranges from  $-1$  to  $1$  (worst to best values) with values near  $0$  means we have many overlapping clusters.
- Thus, the overall Silhouette score is the average Silhouette scores for all samples.

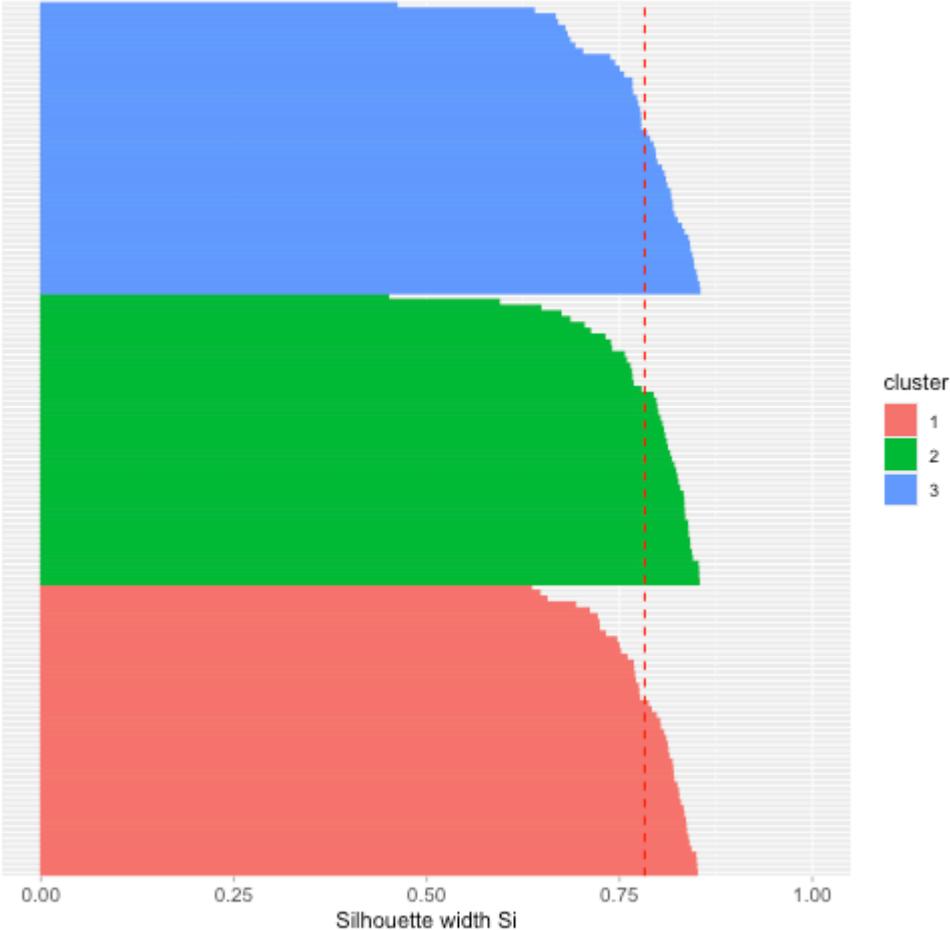
# How do we use Silhouette scores to select the number of clusters?

Clusters silhouette plot  
Average silhouette width: 0.79



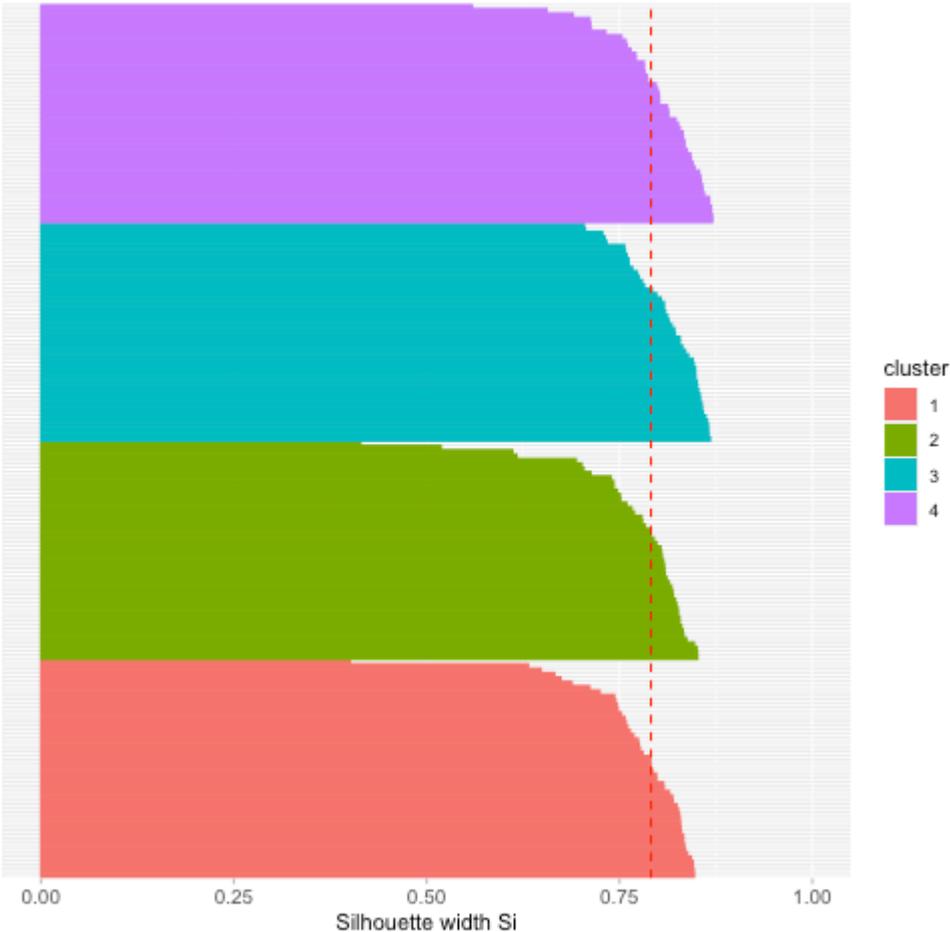
# How do we use Silhouette scores to select the number of clusters?

Clusters silhouette plot  
Average silhouette width: 0.78



# How do we use Silhouette scores to select the number of clusters?

Clusters silhouette plot  
Average silhouette width: 0.79



# Interpretation of Silhouette Plots

- Silhouette Values:
  - Higher silhouette values indicate that a data point is close to its own cluster and far from neighboring clusters, suggesting well-separated clusters.
- Cluster Size Representation:
  - The width of each silhouette reflects the number of data points in the cluster, providing insight into cluster sizes.
- Cluster Quality:
  - The shape and length of each silhouette = the quality of the clustering; a more rectangular shape with a slower drop-off indicates that more points are appropriately clustered.

# Recap and Key Takeaways

- The silhouette method can be applied to evaluate the results of different clustering algorithms beyond **K-Means**.
- Remember, Clustering is an **unsupervised** learning.
- **Distance Metrics**: While not extensively covered here, the choice of distance metrics is a crucial factor in clustering, as it influences how similarity between data points is measured.

# Recap and Key Takeaways

- K-Means requires specifying the number of clusters ( $k$ ) in advance.
- Assigns each data point exclusively to one cluster.
- The cluster labels generated are arbitrary and no specific meaning.
- Cluster centroids are computed as the mean of data points within a cluster
- KMeans always converge: however, that may be suboptimal as it's influenced by the initial centroids.
- **Elbow and Silhouette** techniques are commonly used to assess the optimal number of clusters for a given dataset.
- *Applications*: Data exploration, feature engineering, customer segmentation, and document clustering.
- *Interpreting* the results of clustering algorithms often requires manual effort and domain expertise to extract meaningful insights

Thank you