Cluster Analysis

Computer Applications for the Psychological Sciencees

PSYC470

Contents

Introduction to Cluster Analysis

Cluster analysis is form of *unsupervised machine learning.* Unlike supervised learning, where you have a known target variable (like price or species) to predict, clustering operates in a world without pre-existing labels. The primary goal of cluster analysis is to discover natural groupings or clusters within your data.

The principle is simple:

- Objects within a cluster should be highly similar to one another.
- Objects between different clusters should be highly dissimilar from one another.

Think of it as the computational equivalent of sorting a mixed bag of laundry. You don't have labels telling you what's a sock and what's a shirt, but by looking at features like size, shape, and material, you can intuitively group similar items together. In data analysis, these "features" are your variables, and the "groups" are the hidden patterns you want to uncover.

Cluster analysis is incredibly powerful and widely used for:

- **Customer Segmentation:** Finding distinct groups of customers based on purchasing behavior or demographics.
- **Anomaly Detection:** Identifying data points that don't fit into any cluster, which could represent fraud, errors, or rare events.
- **Image Segmentation:** Grouping pixels in an image to identify objects.
- **Genomics:** Classifying genes based on their expression patterns.
- **Identifying Psychiatric Subtypes:** Many psychiatric diagnoses are very broad and cover people with different combinations of symptoms.
- **Creating Personality Profiles:**
- **Understanding Cognitive Profiles:**

- **Grouping Social Behaviors or Attitudes**

Data Preparation

Before you can run any clustering algorithm (or any other analysis), you must prepare your data. This is arguably the most important step, as most clustering methods are highly sensitive to the scale and nature of your variables.

Feature Scaling

Most clustering algorithms are "distance-based," meaning they calculate the similarity between data points using a mathematical distance.

The Problem: Imagine you have a dataset with two features: age (ranging from 20-70) and income (ranging from \$30,000-\$150,000). When an algorithm calculates the distance, the income variable, simply because of its larger numbers, will completely dominate the calculation. The age variable will have almost no influence on the outcome.

The Solution: You must scale or standardize your data so that all variables contribute equally. The most common method is Z-score standardization, which rescales each variable to have a mean of 0 and a standard deviation of 1.

In R, this is easily done with the `scale()` function:

Calculating Distance ("Similarity")

Once your data is scaled, the algorithm needs a precise mathematical rule to define how "similar" or "dissimilar" two data points are. This rule is the *distance metric*.

Choosing a metric is an important decision, as different metrics can lead to different cluster results. The metric you choose should align with your data's properties and your research question.

Euclidean Distance: This is the most common and intuitive metric. It's the "as the bird flies" straight-line distance between two points. For two data points, $p = (p_1, p_2, \dots, p_n)$ and $q = (q_1, q_2, \dots, q_n)$, across n features (dimensions), the formula is:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

For example, if $n = 2$, then $d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$

Example: Imagine two patients plotted on scaled scores for anxiety (x-axis) and depression (y-axis):

- Patient A: $(p_1, p_2) = (1, 2)$
- Patient B: $(q_1, q_2) = (4, 6)$
- The Euclidean distance is:

$$d(A, B) = \sqrt{(4-1)^2 + (6-2)^2} = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5.0$$

* **Pros:** Very intuitive, represents the shortest physical distance, and works very well for creating compact, spherical clusters. It's the default distance metric for many K-Means algorithms. * **Cons:** Highly sensitive to outliers. Because the differences are squared in the formula, a single large difference on one feature (an outlier) can dramatically inflate the total distance, potentially pulling the point out of its "correct" cluster.

Manhattan Distance: This is often called the "city block" or "taxicab" distance. It calculates the distance by summing the absolute differences of the coordinates, as if you were traveling on a grid of streets. For the same two points, p and q from before, the formula is:

$$d(p,q) = \sum_{i=1}^{n} |p_i - q_i|$$

For example, using the same Patient A and Patient B from before:Patient A: $(p_1, p_2) = (1, 2)$, Patient B: $(q_1, q_2) = (4, 6)$. The Manhattan distance is:

$$d(A, B) = |4-1| + |6-2| = |3| + |4| = 3 + 4 = 7.0$$

Notice the distance (7.0) is different from the Euclidean distance (5.0).

- **Pros:** Much more robust to outliers. Because it uses the absolute difference, not the squared difference, a single large outlier's influence is not exaggerated. It's a good choice if your data is noisy.
- **Cons:** It's less intuitive and can sometimes lead to different (often square-shaped) clusters compared to Euclidean.

Let's see how we can use these two distance metrics to create a distance Matrix in R using the `dist()` function from the base stats package. It takes your scaled data frame (or matrix) and returns a special "dist" object, which is a distance matrix.This matrix contains the calculated distance between every possible pair of data points.Let's create a tiny scaled dataset for 4 patients with anxiety and depression scores.

```
# Create a small, simple data frame
patient_demo <- data.frame(
  anxiety = c(1, 4, 1, 3),
  depression = c(2, 6, 3, 5)
)
rownames(patient_demo) <- c("Patient_A", "Patient_B", "Patient_C", "Patient_D")

print("Original Data:")
```

```
## [1] "Original Data:"
```

```
print(patient_demo)
```

```
##           anxiety depression
## Patient_A       1          2
## Patient_B       4          6
## Patient_C       1          3
## Patient_D       3          5
```

```
# --- 1. Scale the data ---
patient_demo_scaled=scale(patient_demo)

# --- 2. Calculate Euclidean distance matrix ---
dist_euclidean <- dist(patient_demo_scaled, method = "euclidean")

print("--- Euclidean Distance Matrix ---")
```

```
## [1] "--- Euclidean Distance Matrix ---"
```

```
print(dist_euclidean)
```

```
##           Patient_A Patient_B Patient_C
## Patient_B 2.9664794
## Patient_C 0.5477226 2.5884358
## Patient_D 2.1160760 0.8628119 1.7256239
```

```r
# --- 3. Calculate Manhattan distance matrix ---
dist_manhattan <- dist(patient_demo_scaled, method = "manhattan")

print("--- Manhattan Distance Matrix ---")
```

```
## [1] "--- Manhattan Distance Matrix ---"
```

```r
print(dist_manhattan)
```

```
##           Patient_A Patient_B Patient_C
## Patient_B 4.1908902
## Patient_C 0.5477226 3.6431677
## Patient_D 2.9765010 1.2143892 2.4287784
```

Common Types of Clustering Algorithms

There are many clustering methods, but they generally fall into three main families.

1. Partitioning Clustering (e.g., K-Means)

- This method aims to "partition" the data into a pre-specified number of clusters, K. The most popular example is K-Means.
- How it works: You tell the algorithm how many clusters (K) you want to find. It then randomly assigns K points as "centroids" (cluster centers) and iteratively repeats two steps:
  1) Assignment Step: Assigns each data point to its nearest centroid.
  2) Update Step: Recalculates the centroid of each cluster based on the mean of the points assigned to it.
- Best for: General-purpose clustering, fast on large datasets, and tends to create spherical, evenly-sized clusters.

2. Hierarchical Clustering

- This method builds a tree-like hierarchy of clusters, which is visualized as a "dendrogram". It doesn't require you to specify the number of clusters beforehand.
- How it works (Agglomerative): This is the "bottom-up" approach.
  1) Start by treating every single data point as its own cluster.
  2) Find the two "closest" clusters and merges them.
  3) Repeat #2 until all data points are in a single giant cluster. You then "cut" the dendrogram at a certain height to get your desired number of clusters.
- Best for: Smaller datasets where you want to visualize the hierarchy and don't know the number of clusters in advance.

3. Density-Based Clustering (DBSCAN)

- This method groups points that are closely packed together, marking points that lie alone in low-density regions as outliers. The most popular example is DBSCAN (Density-Based Spatial Clustering of Applications with Noise).
- How it works: It defines clusters as continuous regions of high density. It's great at finding arbitrarily-shaped clusters (e.g., rings, curves) and is very robust to outliers.
- Best for: Finding non-spherical clusters, handling noise and outliers effectively.

Cluster Analysis in R

Before going further, let's take a look at how to implement several types of cluster analysis in R.

Data: We will use a real, built-in R dataset for this walkthrough: `USArrests`.

The USArrests dataset is a data frame with 50 rows (one for each US state) and 4 variables: * Murder (arrests per 100,000 residents) * Assault (arrests per 100,000 residents) * UrbanPop (% of the population in urban areas) * Rape (arrests per 100,000 residents)

In this example, we're not clustering individuals, but states. The goal is to find out if there are "profiles" or "types" of states that share similar patterns of crime and urbanization.

First, let's load some packages and prepare our data. This is the most important step. The USArrests data is on different scales (e.g., Assault ranges from 45 to 337, while Murder ranges from 0.8 to 17.4) so we must scale it first.

```r
# --- Load Libraries ---
library(tidyverse)    # For data manipulation and plotting
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(factoextra)   # For cluster visualization & validation
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library(dbscan)       # For the DBSCAN algorithm
```

```
##
## Attaching package: 'dbscan'
##
## The following object is masked from 'package:stats':
##
##     as.dendrogram
```

```r
# --- Load and Inspect Data ---
data("USArrests")
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

```r
# --- Scale the Data ---
# We must scale the data before calculating distances
# This creates 'df' which is a scaled matrix
df <- scale(USArrests)

# View the first few rows of the scaled data
# Notice the means are now 0 and SDs are 1
head(df)
```

```
##                Murder   Assault   UrbanPop         Rape
## Alabama    1.24256408 0.7828393 -0.5209066 -0.003416473
## Alaska     0.50786248 1.1068225 -1.2117642  2.484202941
## Arizona    0.07163341 1.4788032  0.9989801  1.042878388
## Arkansas   0.23234938 0.2308680 -1.0735927 -0.184916602
## California 0.27826823 1.2628144  1.7589234  2.067820292
## Colorado   0.02571456 0.3988593  0.8608085  1.864967207
```

Now df is our prepared, scaled data, ready for clustering.

Method 1: K-Means Clustering

Goal: Partition the 50 states into K groups…but first we have to find the best K.

How Many Clusters?

For methods like K-Means, how do you pick the "right" number for K? There is no single, correct answer. Instead, we use heuristics to find an "optimal" number.

1) The Elbow Method: This method plots the Within-Cluster Sum of Squares (WCSS) against different values of K. The WCSS measures the compactness of the clusters. As K increases, WCSS will always decrease. You look for the "elbow" in the plot—the point where adding another cluster doesn't significantly reduce the WCSS.

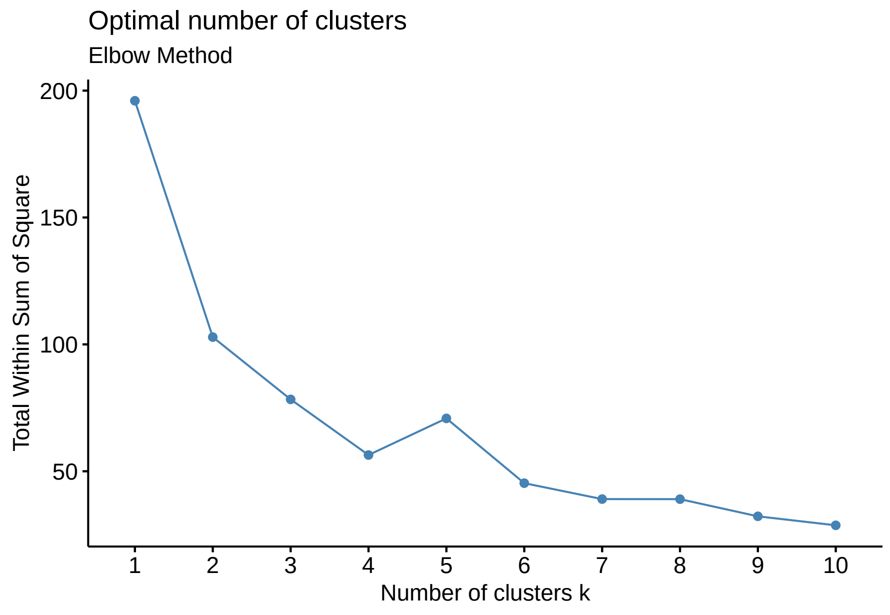
2) The Silhouette Score: This method provides a more robust measure. For each data point, it calculates a score based on two things:

- Cohesion: How close is it to other points in its own cluster?
- Separation: How far is it from points in the nearest other cluster?

The score ranges from +1 (perfectly clustered) to -1 (wrongly clustered). You calculate the average silhouette score for different values of K and pick the K that maximizes the average score.

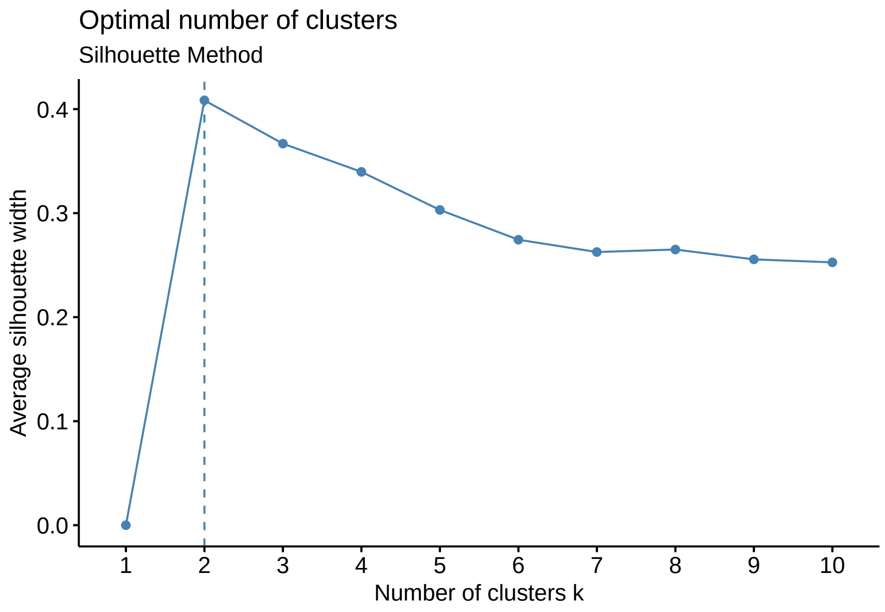
The `factoextra` package makes this simple.

```r
# Find optimal K using the Elbow method
fviz_nbclust(df, kmeans, method = "wss") +
  labs(subtitle = "Elbow Method")
```



Interpretation: The plot shows a very clear "elbow" at K=4. After this point, adding more clusters doesn't significantly reduce the within-cluster sum of squares (WCSS).

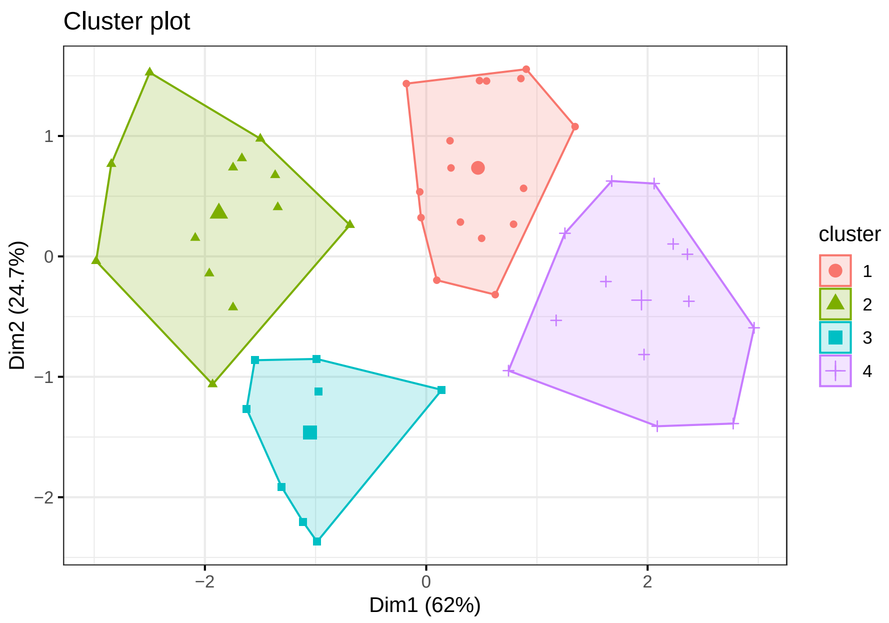
```
# Find optimal K using the Silhouette method
fviz_nbclust(df, kmeans, method = "silhouette") +
  labs(subtitle = "Silhouette Method")
```



Interpretation: The silhouette method suggests K=2 is the most optimal, as it has the highest score. However, $K = 4$ was indicated using the elbow method. This is a common "conflict" in clustering. $K = 2$ would probably just separate low-crime/rural from high-crime/urban states. $K = 4$ will likely give us more nuanced profiles. For this educational example, let's choose K=4 to get more interesting groups.

Run K-means algorithm with our chosen $K = 4$.

```
# Set seed for reproducibility
set.seed(123)

# Run K-Means
# nstart=25 will run the algorithm 25 times with different
# random starting points and pick the best one.
km.res <- kmeans(df, centers = 4, nstart = 25)

# Visualize the clusters
# fviz_cluster will perform PCA (Principle Component Analysis)
# to show the 4-dimensional data in 2 dimensions.
fviz_cluster(km.res, data = df,
             ellipse.type = "convex",
             geom = "point",
             ggtheme = theme_bw())
```



Interpretation: When we ran the K-Means algorithm on the USArrests data, we used four variables: Murder, Assault, UrbanPop, and Rape. This means both our data, and the $k = 4$ clusters we found, exist in 4-dimensional space.

A data point for any given state isn't just (x, y). It's (x, y, z, w), or more specifically: (Murder_score, Assault_score, UrbanPop_score, Rape_score).

The center of each cluster (the centroid) is also a 4-dimensional point.

This creates a visualization problem: our computer screens are 2-dimensional. How can we possibly plot 4D data on a 2D screen?

A "Bad" Solution: You might think, "Why not just plot two of the original variables, like Murder vs. Assault?"

The Problem: You'd be ignoring all the information from UrbanPop and Rape. It's possible that two clusters (e.g., Cluster 1 and Cluster 3) look identical on the Murder/Assault plot, but are actually very different, separated only by the UrbanPop variable. You would be looking at a misleading "shadow" of the data.

The "Smart" Solution: Principal Component Analysis (PCA) PCA is a dimensionality reduction technique. Its goal is to take a high-dimensional dataset (like our 4D data) and "squish" or "project" it onto a lower number of dimensions (like 2D) while losing as little information as possible.

An Analogy: Imagine you have a complex 3D object, like a model airplane. You want to take a 2D photograph of it to show someone. If you take the photo from directly in front of the plane, your 2D image will just be a thin "cross" shape. You've lost all the information about the wings' length and the plane's body.

However, if you rotate the plane to a 3/4 angle from above, your 2D photo will capture the shape of the wings, the body, and the tail. This is a much more informative "shadow" or "projection" of the 3D object.

PCA is the mathematical method for finding this "best angle" in high-dimensional data.

How PCA Works (Step-by-Step) 1. Finding New Axes (The "Principal Components") * PCA doesn't just pick two of our old axes (like Murder and Assault). Instead, it creates brand new, artificial axes called Principal Components (PCs). * These new axes are linear combinations of the old ones. * *Principal Component 1 (PC1):* PCA finds the "best" possible new axis that captures the maximum amount of variance (the "spread") in the 4D data. - It's a formula like: PC1 = (a * Murder) + (b * Assault) + (c * UrbanPop) + (d * Rape). PCA finds the perfect "weights" (a, b, c, d) that make the 50 states as spread out as possible along this new line. This is our "best angle" for the shadow. * *Principal Component 2 (PC2):* PCA then finds the second-best new axis. This axis must be orthogonal (perpendicular) to PC1 and must capture the most of the remaining variance. It has its own formula: PC2 = (e * Murder) + (f * Assault) + (g * UrbanPop) + (h * Rape) * This process continues until we have 4 PCs (PC1, PC2, PC3, PC4), one for each original variable.

2. Checking if the New Axes are "Good Enough"

- The magic of PCA is that the first few components capture most of the information. We can check how much "variance" (information) PC1 and PC2 captured.
- PC1 might capture (for example) 62% of the total variance from all 4 original variables.
- PC2 might capture (for example) 25% of the total variance.
- This means our new 2D plot (PC1 vs. PC2) successfully represents 62% + 25% = 87% of the original 4D information! This is far better than just plotting Murder vs. Assault, which might only represent 50% (or less) of the total variance.

This is exactly what the `fviz_cluster()` function does for us automatically:

It ignores the original variables (Murder, Assault, etc.).

It runs PCA on your 4D scaled data (df) to find the "best" new axes, PC1 and PC2.

It calculates the new coordinates for all 50 states. For example, Alabama is no longer (-0.99, -0.15, -1.22, 0.51) in 4D space. It's now something like (-0.98, 1.13) in the new 2D (PC1/PC2) space.

It draws a 2D scatter plot with these new coordinates:

X-axis = PC1 score ("Dimension 1")

Y-axis = PC2 score ("Dimension 2")

It uses your K-Means results (the km.res$cluster vector) to color the points. It looks at Alabama, sees it belongs to cluster = 1, and colors its point red. It looks at California, sees cluster = 2, and colors it green.

Summary

The fviz_cluster plot is not plotting any of your original variables.

It is a "smart" 2D plot where the X-axis (Dim 1) and Y-axis (Dim 2) are new, artificial variables (PC1 and PC2) created by PCA. These new variables are combinations of all four original variables, designed to give you the best possible 2D view of the separation between your 4-dimensional clusters.

Interpret the K-Means Profiles

Now let's find out what these clusters mean by looking at their average scores. We'll add the cluster assignments back to the original, unscaled data.

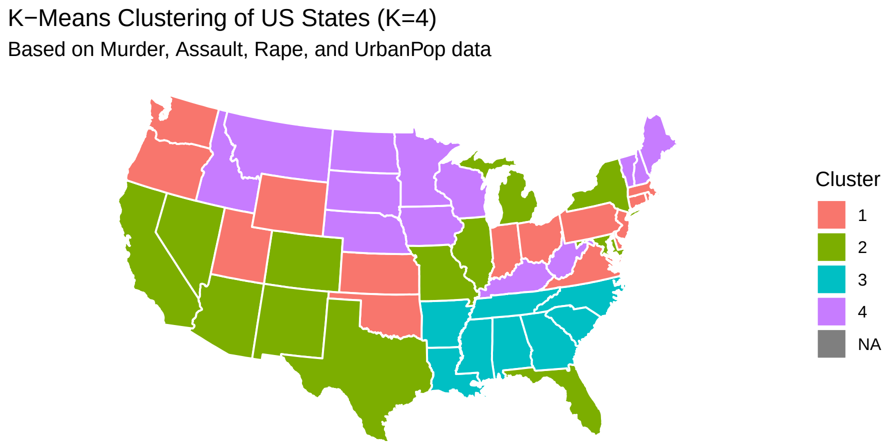
```r
# Add cluster assignments to the original data
USArrests_kmeans <- USArrests %>%
  mutate(cluster = km.res$cluster)

# Calculate the mean of each variable for each cluster
cluster_profiles_km <- USArrests_kmeans %>%
  group_by(cluster) %>%
  summarise(
    n = n(),
    Murder_mean = mean(Murder),
    Assault_mean = mean(Assault),
    Rape_mean = mean(Rape),
    UrbanPop_mean = mean(UrbanPop)
  )

print(cluster_profiles_km)
```

```
## # A tibble: 4 x 6
##   cluster     n Murder_mean Assault_mean Rape_mean UrbanPop_mean
##     <int> <int>       <dbl>        <dbl>     <dbl>         <dbl>
## 1       1    16        5.66         139.      18.8          73.9
## 2       2    13       10.8          257.      33.2          76
## 3       3     8       13.9          244.      21.4          53.8
## 4       4    13        3.6           78.5     12.2          52.1
```

Interpretation: * Cluster 1: High urban population but relatively low crime rate. * Cluster 2: High urban population with high crime rate. * Cluster 3: Low urban population with higher crime rate. * Cluster 4: Low urban population with lower crime rate.

This is a great place for a chorograph!

```r
# --- 3. Prepare Cluster Data for Merging ---
# Create a data frame with state names and their cluster
# Note: The map data uses *lowercase* state names, so we must convert them
cluster_data <- data.frame(
  region = tolower(rownames(USArrests)),
  cluster = as.factor(km.res$cluster) # Convert cluster number to a factor
)

# --- 4. Load the US Map Polygon Data ---
# This function from ggplot2 converts the 'maps' data into a data frame
us_map <- map_data("state")

# --- 5. Join Your Clusters to the Map Data ---
# We join the map polygons (us_map) with our cluster assignments (cluster_data)
# The 'by = "region"' tells R to match the 'region' column in both datasets
map_with_clusters <- left_join(us_map, cluster_data, by = "region")

# --- 6. Plot the Final Map ---
ggplot(map_with_clusters, aes(x = long, y = lat, group = group)) +
  # Create the polygons
  # 'fill = cluster' colors the states based on our cluster assignment
  # 'color = "white"' adds a thin white border between states
  geom_polygon(aes(fill = cluster), color = "white") +

  # Use a standard map projection (Albers is good for the US)
  coord_map("albers", lat0 = 39, lat1 = 45) +

  # Add titles
  labs(
    title = "K-Means Clustering of US States (K=4)",
    subtitle = "Based on Murder, Assault, Rape, and UrbanPop data",
    fill = "Cluster" # This renames the legend title
  ) +

  # Use a clean theme that removes axes and gridlines
  theme_void()
```


K−Means Clustering of US States (K=4)
Based on Murder, Assault, Rape, and UrbanPop data
Cluster
1
2
3
4
NA

Method 2: Hierarchical Clustering

Goal: Build a "family tree" (dendrogram) of states to see how they group from the bottom up. We don't need to pre-specify K.

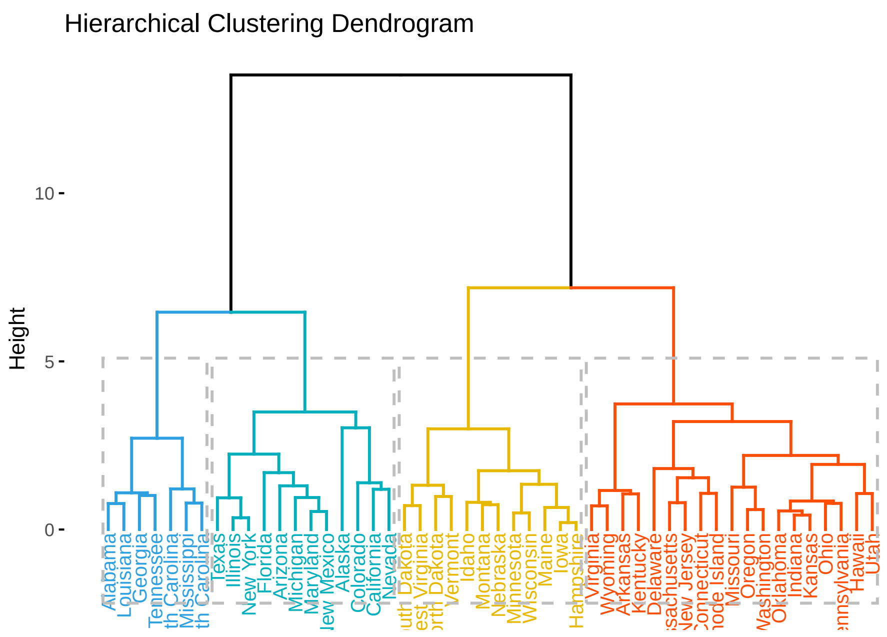
A. Calculate Distance & Run Algorithm * This method needs a distance matrix as its input. * We'll use the dist() function on our scaled data.

```
# 1. Calculate the distance matrix
d <- dist(df, method = "euclidean")

# 2. Run hierarchical clustering
# "ward.D2" is a popular method that minimizes within-cluster variance
hc.res <- hclust(d, method = "ward.D2")
```

B. Visualize the Dendrogram * The primary output is the dendrogram, which shows the hierarchy.

```
# 3. Plot the dendrogram
fviz_dend(hc.res, cex = 0.7,  # cex = label size
          k = 4, # "Cut" the tree into 4 clusters
          rect = TRUE, # Add rectangles around the 4 clusters
          k_colors = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
          main = "Hierarchical Clustering Dendrogram")
```

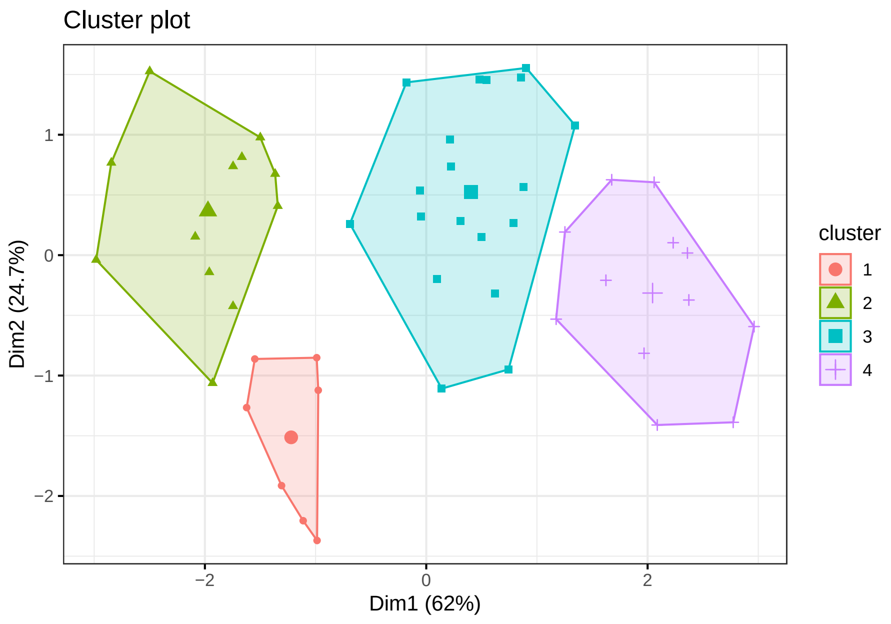
```
## Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none" instead as
## of ggplot2 3.3.4.
## i The deprecated feature was likely used in the factoextra package.
##   Please report the issue at <https://github.com/kassambara/factoextra/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Interpretation: You can "read" the tree from the bottom. Alabama and Louisiana are very similar, so they are the first to "merge." This group then merges with Mississippi, and so on. The long vertical lines show that the four main clusters are very dissimilar from each other.

C. Visualize the Clusters (like K-Means) * We can "cut" the tree at $K = 4$ (to match our K-Means) and visualize the clusters on a scatter plot.

```r
# Cut the tree into 4 groups
clusters_hc <- cutree(hc.res, k = 4)

# Visualize the clusters on a scatter plot
fviz_cluster(list(data = df, cluster = clusters_hc),
             ellipse.type = "convex",
             geom = "point",
             ggtheme = theme_bw())
```



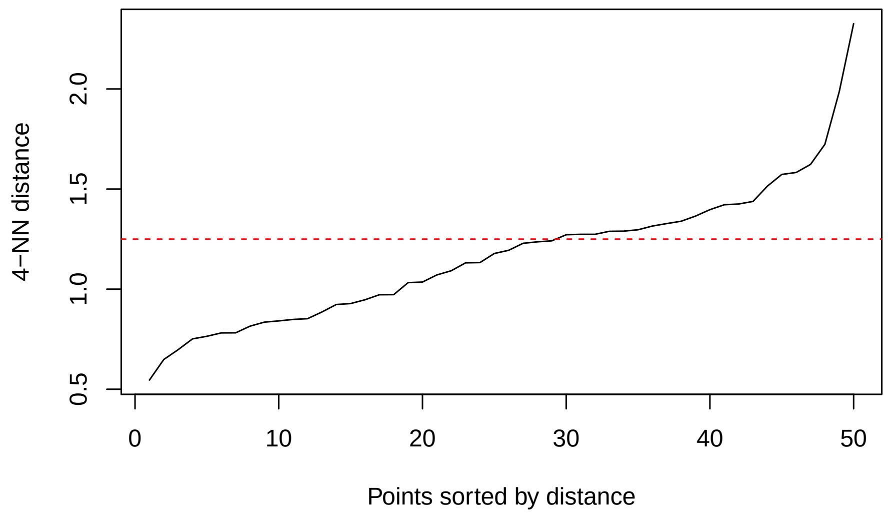
Interpretation: Notice the results are similar, but not identical to K-Means. This is because the logic of how the clusters are formed is different (bottom-up hierarchy vs. top-down partitioning).

Method 3: DBSCAN (Density-Based)

Goal: Find clusters based on density. This method can also identify "outliers" or "noise"—states that don't fit into any profile.

A. Find the Optimal eps ParameterDBSCAN doesn't need K, but it does need an eps (epsilon) value, which is the "neighborhood radius" to search. The standard way to find it is to plot the distance of every point to its k-nearest neighbors (we'll use $k = 4$ neighbors) and look for the "knee" or "elbow" in the plot.

```r
# minPts = k+1. If we want k=4 neighbors, minPts = 5.
# This plots the distance to the 4th nearest neighbor for all 50 states
kNNdistplot(df, k = 4)
abline(h = 1.25, lty = 2, col = "red") # lty=2 is a dashed line
```



Interpretation: The "knee" of the plot is at the point where the distances start to rise sharply. This looks to be around eps = 1.25. We'll use this value. We will also set minPts = 5 (a common default, $k + 1$).

B. Run DBSCAN and Visualize

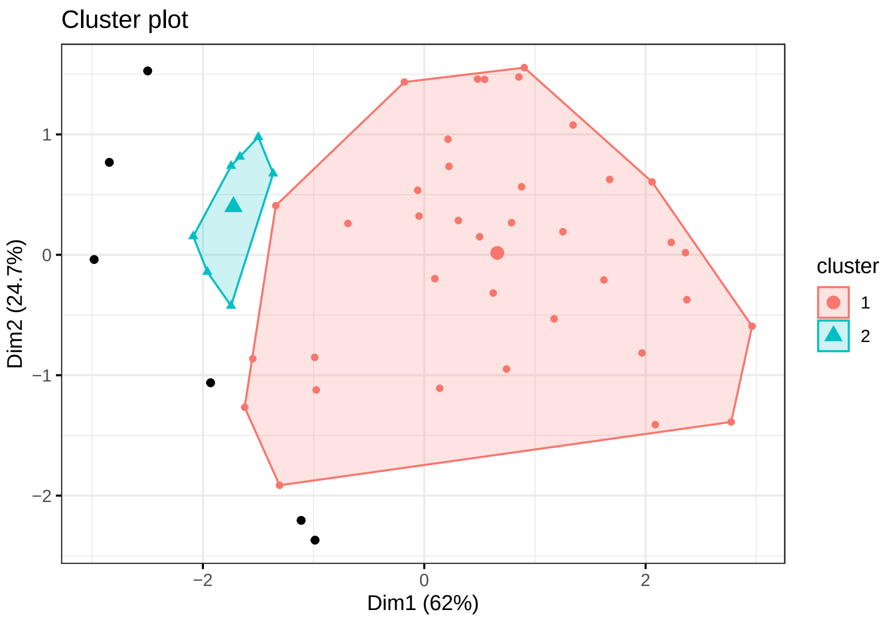
```r
# Run DBSCAN
db.res <- dbscan(df, eps = 1.25, minPts = 5)

# Print the results
print(db.res)
```

```
## DBSCAN clustering for 50 objects.
## Parameters: eps = 1.25, minPts = 5
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 2 cluster(s) and 6 noise points.
##
##  0  1  2
##  6 37  7
##
## Available fields: cluster, eps, minPts, metric, borderPoints
```

Interpretation: This is fascinating! DBSCAN found 2 clusters (labeled 1, 2) and 6 noise points (labeled 0). This means it identified 6 states as being "outliers" that don't fit well with any other group.

C. Visualize the DBSCAN Clusters

```r
# Visualize, note how '0' (black points) are the outliers
fviz_cluster(db.res, data = df,
             ellipse = TRUE,
             geom = "point",
             ggtheme = theme_bw())
```



Final Interpretation: This might be the most "honest" result. K-Means and Hierarchical forced every state into a group. DBSCAN tells us that states like Florida, Alaska, California, and Nevada (which are often in the "noise" cluster) might be too unique to be grouped, and we are left with two more homogenous, tightly-packed clusters.

References

Hartigan, JA, and MA Wong. 1979. "Algorithm AS 136: A K-means clustering algorithm." Applied Statistics. Royal Statistical Society, 100–108.

MacQueen, J. 1967. "Some Methods for Classification and Analysis of Multivariate Observations." In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, 281–97. Berkeley, Calif.: University of California Press. http://projecteuclid.org:443/euclid.bsmsp/1200512992.