Learning K-Medoids Clustering with a Step-by-Step Example in R

Authored by

Mohammed looti

November 6, 2025

Clustering is a fundamental technique in **machine learning** used to identify inherent groupings, or clusters, of data points within a dataset. The core objective is to ensure that observations within any single cluster are highly similar to each other, while remaining distinctly different from observations in other clusters.

Since clustering seeks to discover underlying structure rather than predict a known outcome, it falls under the category of unsupervised learning. This method is highly valuable in fields like marketing, where it helps segment customers based on shared characteristics.

Typical data used for clustering in business contexts includes information on:

Household income demographics
Household size and composition
Occupation of the head of household
Geographic metrics, such as distance from urban centers

By analyzing this information, clustering allows organizations to pinpoint similar households, enabling targeted advertising and product recommendations. While K-means clustering is perhaps the most widely known method, it is sensitive to extreme values. A robust and highly effective alternative is **K-medoids clustering**.

Understanding K-Medoids Clustering

K-medoids clustering, often referred to as Partitioning Around Medoids (PAM), is a partitioning method designed to enhance robustness compared to K-means. Like K-means, the goal is to partition the dataset into *K* distinct clusters.

The key difference lies in how the cluster center is defined. Instead of using a calculated mean (which might not correspond to an actual data point), K-medoids uses an actual data point--the **medoid**--as the cluster center. The medoid is the most centrally located data point within a cluster.

Because it relies on medians and actual observations rather than means, K-medoids is significantly less susceptible to the distorting influence of outliers, making it a powerful choice for datasets prone to extreme values.

The K-Medoids Algorithm: Step-by-Step Process

The implementation of K-medoids clustering follows a systematic iterative procedure aimed at minimizing the distance between data points and their assigned medoids. We can summarize the algorithm in the following three primary steps:

Initialization: Select the value for *K*. The user must first define the desired number of clusters, *K*.

Since the optimal *K* is often unknown, techniques such as the Elbow method or Gap statistic (discussed later) are typically employed to assess multiple values and determine the most appropriate number of clusters for the given data structure.

Medoid Selection: Choose initial medoids. Randomly select *K* observations from the dataset to serve as the initial medoids (cluster centers). Each remaining observation is then assigned to its nearest medoid.

Iteration and Swapping: Optimize cluster assignment. The algorithm iteratively performs the following until cluster assignments stabilize:

Calculate the total cost of the current configuration (typically the sum of dissimilarities between points and their medoids).

Attempt to swap a chosen medoid with a non-medoid point.

If the swap reduces the overall cost, the swap is accepted, and the procedure repeats. If the swap increases the cost, the original medoid is retained.

The distance metric used to define 'closest' in this context is often the Euclidean distance, though other metrics like Manhattan distance can be employed depending on the nature of the data.

Technical Note: Robustness to Outliers

The superior robustness of K-medoids stems from its use of **medians** (represented by the medoid) rather than means when defining the cluster center. Means are highly sensitive to extreme data points, whereas medians are not.

In datasets where outliers are sparse or nonexistent, K-means and K-medoids typically yield similar clustering results. However, K-medoids offers a distinct advantage when data quality is a concern.

Implementing K-Medoids in R: Setup and Data Preparation

To demonstrate K-medoids clustering, we will utilize the R programming environment. The primary function for this technique is `pam()` (Partitioning Around Medoids), which is housed within the `cluster` package. We also need the `factoextra` package for visualization and optimal cluster determination.

We begin by loading the required libraries:

```
library(factoextra)
library(cluster)
```

For our practical example, we will employ the built-in R dataset `USArrests`. This dataset documents arrest rates per 100,000 residents across all U.S. states in 1973 for the crimes of *Murder*, *Assault*, and *Rape*, alongside the percentage of the population living in urban areas (*UrbanPop*).

Before running any clustering algorithm, it is critical to preprocess the data. The following steps ensure our data is clean and appropriately scaled:

Loading the `USArrests` dataset into a working dataframe.
Handling missing values (NAs) by removing affected rows.
Standardizing all variables to have a mean of 0 and a standard deviation of 1. Scaling is vital for distance-based clustering methods to prevent variables with larger magnitudes from unfairly dominating the distance calculations.

 # Load and clean data
df <- USArrests

Remove rows with missing values
df <- na.omit(df)

Scale variables (standardization)
df <- scale(df)

# View the scaled data structure
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

Determining the Optimal Number of Clusters (K)

A critical step in K-medoids is selecting the optimal number of clusters, *K*. In R, we use the `pam()` function, which takes the standardized data, the desired *K*, and optional parameters for distance calculation.

The general syntax for the `pam()` function is: **pam(data, k, metric = "euclidean", stand =**

FALSE). The primary parameters include `data` (the matrix or dataframe to cluster), `k` (the specified number of clusters), and `metric` (the distance function, defaulting to Euclidean distance).

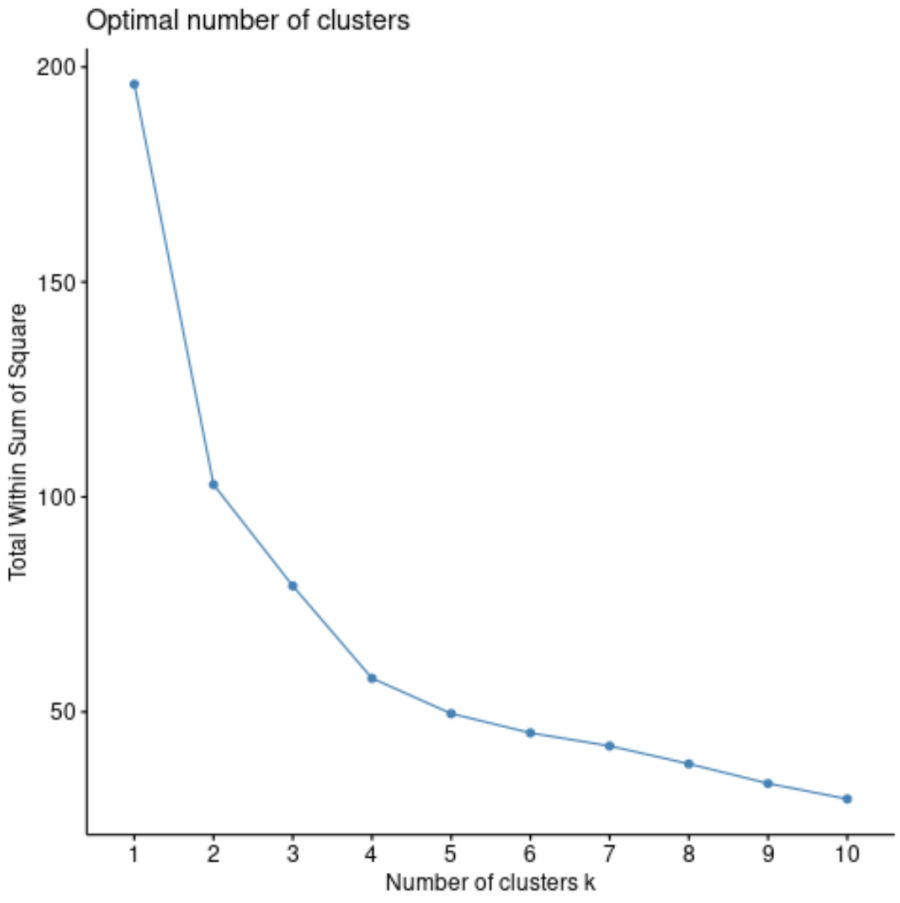
Since *K* is unknown, we employ two standard diagnostic methods to guide our selection: the Elbow Method (WSS) and the Gap Statistic.

Method 1: The Elbow Method (Total Within Sum of Squares)

The Elbow Method plots the number of clusters against the Total Within Sum of Squares (WSS). The WSS measures the compactness of the clustering--a smaller WSS value indicates tighter clusters. We expect WSS to decrease as *K* increases, but we look for the point where the rate of decrease significantly slows down, forming an "elbow." This bend often suggests the optimal trade-off between minimizing WSS and avoiding unnecessary complexity.

We use the `fviz_nbclust()` function for visualization:

fviz_nbclust(df, pam, method = "wss")



Analyzing the plot, we observe a noticeable bend at *k* = 4 clusters. Selecting a *K* value far beyond

this elbow might lead to overfitting, where the model captures noise rather than meaningful structure.

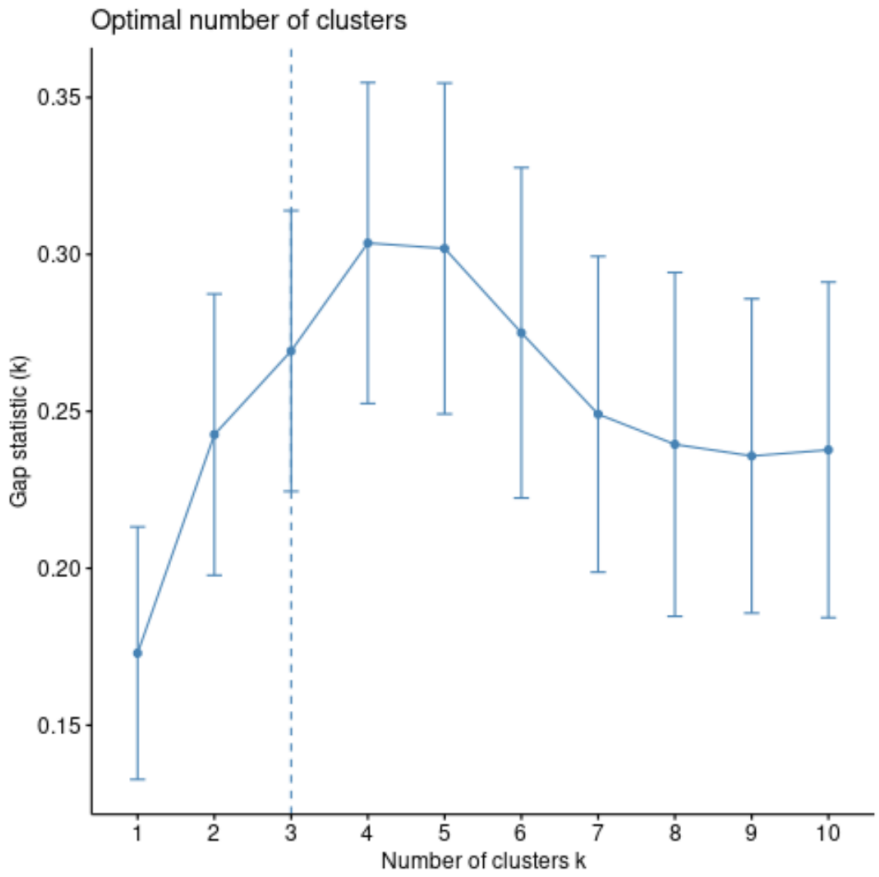
Method 2: The Gap Statistic

The Gap statistic provides a more formal way to estimate *K*. It compares the total intra-cluster variation of the observed data to that of a reference null distribution (a dataset with no inherent clustering). The optimal *K* is typically the value that maximizes the gap statistic.

We first calculate the statistic using `clusGap()` and then visualize the results with `fviz_gap_stat()`:

```
# Calculate gap statistic based on number of clusters
gap_stat <- clusGap(df,
FUN = pam,
K.max = 10, # Maximum clusters to evaluate
B = 50) # Total bootstrapped iterations

# Plot number of clusters vs. gap statistic
fviz_gap_stat(gap_stat)
```



Consistent with the Elbow Method, the Gap statistic plot confirms that the highest value occurs at *k* = 4 clusters. We will proceed with this value for our final model.

Executing and Interpreting the Final K-Medoids Model

Having confirmed that *K*=4 is the optimal cluster number using both the WSS and Gap statistic methods, we now execute the K-medoids algorithm using the `pam()` function on our standardized `USArrests` data.

We set a seed for reproducibility and run the model:

```
# Make this example reproducible
set.seed(1)

# Perform K-medoids clustering with k = 4 clusters
kmed <- pam(df, k = 4)

# View the model results
kmed
```

ID Murder Assault UrbanPop Rape
Alabama 1 1.2425641 0.7828393 -0.5209066 -0.003416473
Michigan 22 0.9900104 1.0108275 0.5844655 1.480613993
Oklahoma 36 -0.2727580 -0.2371077 0.1699510 -0.131534211
New Hampshire 29 -1.3059321 -1.3650491 -0.6590781 -1.252564419
Clustering vector:
Alabama Alaska Arizona Arkansas California
1 2 2 1 2
Colorado Connecticut Delaware Florida Georgia
2 3 3 2 1
Hawaii Idaho Illinois Indiana Iowa
3 4 2 3 4
Kansas Kentucky Louisiana Maine Maryland
3 3 1 4 2
Massachusetts Michigan Minnesota Mississippi Missouri
3 2 4 1 3
Montana Nebraska Nevada New Hampshire New Jersey
3 3 2 4 3
New Mexico New York North Carolina North Dakota Ohio
2 2 1 4 3
Oklahoma Oregon Pennsylvania Rhode Island South Carolina
3 3 3 3 1
South Dakota Tennessee Texas Utah Vermont
4 1 2 3 4
Virginia Washington West Virginia Wisconsin Wyoming
3 3 4 4 3
Objective function:
build swap
1.035116 1.027102

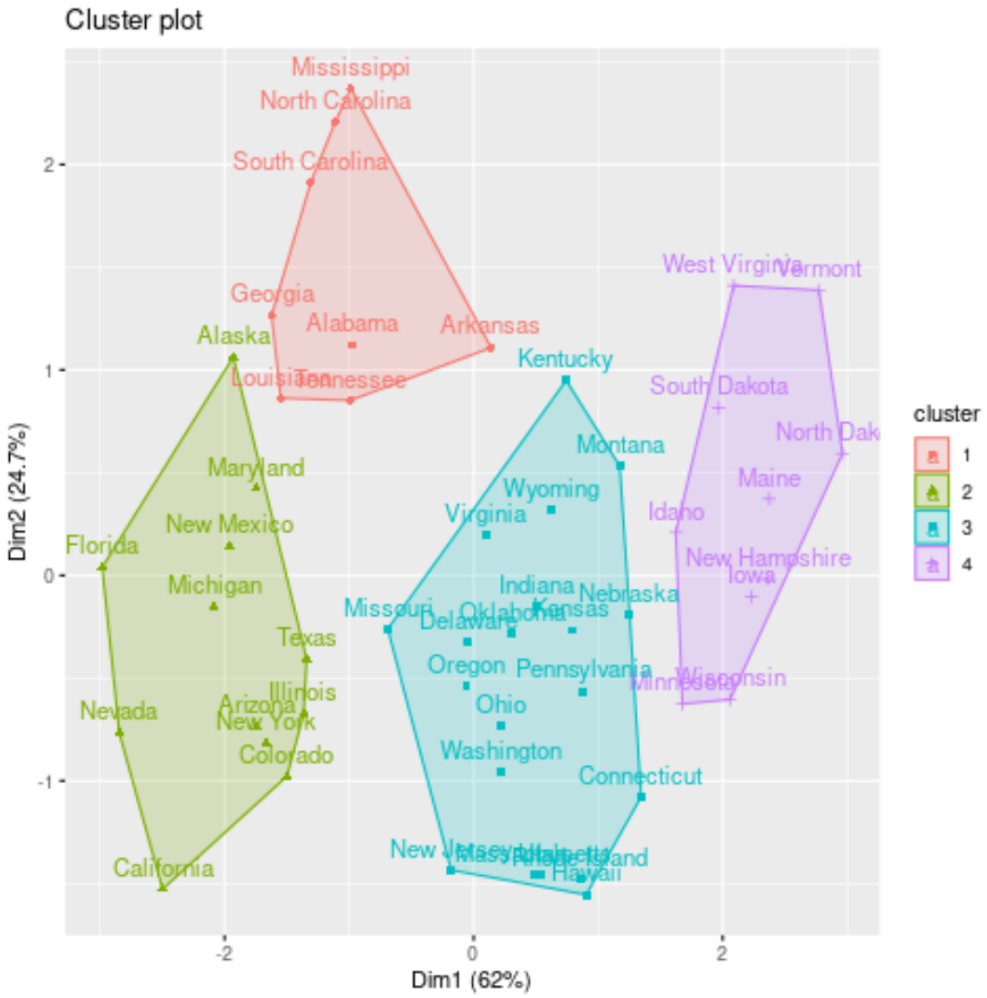
Available components:
"medoids" "id.med" "clustering" "objective" "isolation"
"clusinfo" "silinfo" "diss" "call" "data"

Upon reviewing the output, we confirm that the four cluster centers are defined by actual observations, known as medoids. In this case, the states representing the center of each cluster are: **Alabama** (Cluster 1), **Michigan** (Cluster 2), **Oklahoma** (Cluster 3), and **New Hampshire** (Cluster 4).

To better understand the separation achieved by the model, we can visualize the cluster results.

The `fviz_cluster()` function projects the clusters onto a scatterplot based on the first two principal components, allowing for easy interpretation of group boundaries.

 # Plot results of final K-medoids model
fviz_cluster(kmed, data = df)



Finally, for practical analysis, it is useful to append the determined cluster assignments back to the original, unscaled dataset. This step allows analysts to examine the raw characteristics of the states within each identified cluster.

 # Add cluster assignment to original data
final_data <- cbind(USArrests, cluster = kmed$cluster)

# View the resulting dataframe
head(final_data)

Murder Assault UrbanPop Rape cluster
Alabama 13.2 236 58 21.2 1
Alaska 10.0 263 48 44.5 2
Arizona 8.1 294 80 31.0 2
Arkansas 8.8 190 50 19.5 1
California 9.0 276 91 40.6 2
Colorado 7.9 204 78 38.7 2

The complete R code script used throughout this implementation guide is available for download here.