Clustering in R

Contents

A. Clustering Overview

Clustering is a broad set of techniques for finding subgroups of observations within a data set. When we cluster observations, we want observations in the same group to be similar and observations in different groups to be dissimilar. Because there isn't a response variable, this is an unsupervised method, which implies that it seeks to find relationships between the n observations without being trained by a response variable.

Types of clustering based on categorization of each point:

1. Hard Clustering - In hard clustering, each data point either belongs to a cluster completely or not. For example, in the above example each customer is put into one group out of the 10 groups.
2. Soft Clustering - In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned. For example, from the above scenario each costumer is assigned a probability to be in either of 10 clusters of the retail store.

Read more at https://en.wikipedia.org/wiki/Cluster_analysis

B. Problem Definiton

We are trying to find groups of states that are similar based on the types of crime that is prevalent. Here, we'll use the built-in R data set USArrests, which contains statistics in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states

```r
# Load Packages

library(dplyr)
library(tidyverse)
library(factoextra)
library(cluster)
```

```r
# Load Data

df <- USArrests
head(df)
```

```
##            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
str(df)
```

```
## 'data.frame':    50 obs. of  4 variables:
##  $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
##  $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
##  $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
##  $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
```

```r
summary(df)
```

```
##      Murder          Assault         UrbanPop          Rape      
##  Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30  
##  1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07  
##  Median : 7.250   Median :159.0   Median :66.00   Median :20.10  
##  Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23  
##  3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18  
##  Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00  
```

Based on our initial exploration this dataset is clean, without any missing values, outliers, or apparent mistakes.

B. K-Means

The basic idea behind k-means clustering consists of defining clusters so that the total intra-cluster variation (known as total within-cluster variation) is minimized.

K-means Algorithm

The first step when using k-means clustering is to indicate the number of clusters (k) that will be generated in the final solution. The algorithm starts by randomly selecting k objects from the data set to serve as the initial centers for the clusters. The selected objects are also known as cluster means or centroids. Next, each of the remaining objects is assigned to it's closest centroid, where closest is defined using the Euclidean distance between the object and the cluster mean. This step is called "cluster assignment step". After the assignment step, the algorithm computes the new mean value of each cluster. The term cluster "centroid update" is used to design this step. Now that the centers have been recalculated, every observation is checked again to see if it might be closer to a different cluster. All the objects are reassigned again using the updated cluster means. The cluster assignment and centroid update steps are iteratively repeated until the cluster assignments stop changing (i.e until convergence is achieved). That is, the clusters formed in the current iteration are the same as those obtained in the previous iteration.

K-means algorithm can be summarized as follows:

1. Specify the number of clusters (K) to be created (by the analyst)
2. Select randomly k objects from the data set as the initial cluster centers or means
3. Assigns each observation to their closest centroid, based on the Euclidean distance between the object and the centroid
4. For each of the k clusters update the cluster centroid by calculating the new mean values of all the data points in the cluster. The centroid of a Kth cluster is a vector of length p containing the means of all variables for the observations in the kth cluster; p is the number of variables.
5. Iteratively minimize the total within sum of square. That is, iterate steps 3 and 4 until the cluster assignments stop changing or the maximum number of iterations is reached. By default, the R software uses 10 as the default value for the maximum number of iterations.

K-means needs that data to be normalized.

After inspecting the data, it is obvious that the attributes have different value ranges. The data must be standardized (i.e., scaled) to make variables comparable.

```r
# Normalize

normalize <- function(x){
  return ((x - min(x))/(max(x) - min(x)))}

#df = mutate(df, Murder = normalize(Murder),Assault = normalize(Assault),
#                 UrbanPop = normalize(UrbanPop),Rape = normalize(Rape) )
```

```r
# Standardize

df <- scale(df)
summary(df)
```

```
##      Murder            Assault            UrbanPop             Rape
##  Min.   :-1.6044   Min.   :-1.5090   Min.   :-2.31714   Min.   :-1.4874
##  1st Qu.:-0.8525   1st Qu.:-0.7411   1st Qu.:-0.76271   1st Qu.:-0.6574
##  Median :-0.1235   Median :-0.1411   Median : 0.03178   Median :-0.1209
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.7949   3rd Qu.: 0.9388   3rd Qu.: 0.84354   3rd Qu.: 0.5277
##  Max.   : 2.2069   Max.   : 1.9948   Max.   : 1.75892   Max.   : 2.6444
```

K-means clustering, using the kmeans() function is stats package. The kmeans() function takes the raw data as well as the user-specified k (number of clusters) as input. From its output, we can identify the size of each cluster, the centroid of each cluster, and the cluster assignment of each data point. Using the last piece of information, we can plot a scatterplot matrix visualizing the clustering results.

```r
kcluster <- kmeans(df, 3)
kcluster$size
```

```
## [1] 29 13  8
```

```r
kcluster$centers
```

```
##       Murder    Assault    UrbanPop        Rape
## 1 -0.7010700 -0.7071522 -0.09924526 -0.57773737
## 2  0.6950701  1.0394414  0.72263703  1.27693964
## 3  1.4118898  0.8743346 -0.81452109  0.01927104
```

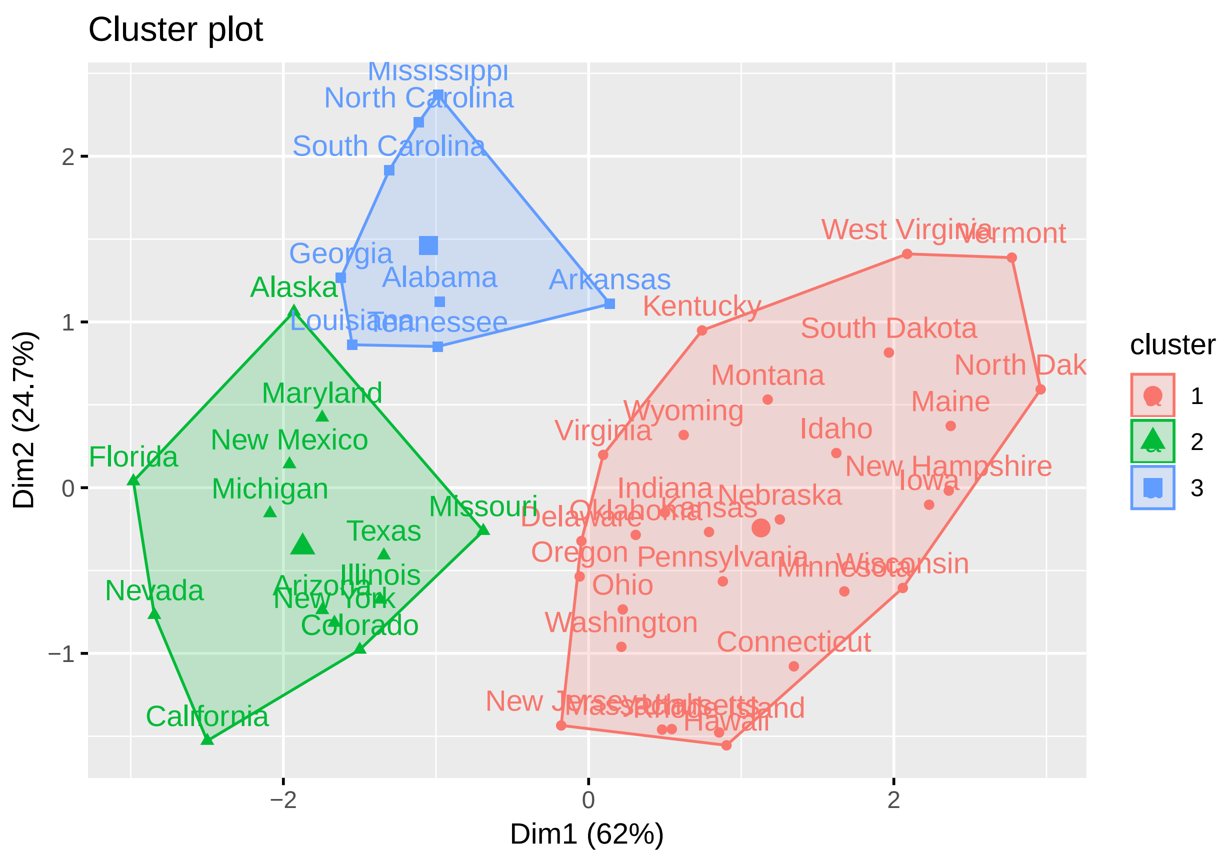
```r
kcluster$cluster
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              3              2              2              3              2
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              2              1              1              2              3
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              1              1              2              1              1
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              1              1              3              1              2
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              1              2              1              3              2
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              1              1              2              1              1
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              2              2              3              1              1
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              1              1              1              1              3
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              1              3              2              1              1
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              1              1              1              1              1
```

If we print the results we'll see that our groupings resulted in 3 cluster sizes of 29, 8 and 13. We see the cluster centers (means) for the three groups across the four variables (Murder, Assault, UrbanPop, Rape). We also get the cluster assignment for each observation (i.e. Alabama was assigned to cluster 2, Arkansas was assigned to cluster 3, etc.).

We can also view our results by using fviz_cluster. This provides a nice illustration of the clusters. If there are more than two dimensions (variables) fviz_cluster will perform principal component analysis (PCA) and plot the data points according to the first two principal components that explain the majority of the variance.

```
fviz_cluster(kcluster, data = df)
```



Determining Optimal Clusters

The three most popular methods for determining the optimal clusters:

1. Elbow method
2. Silhouette method
3. Gap statistic

Elbow Method

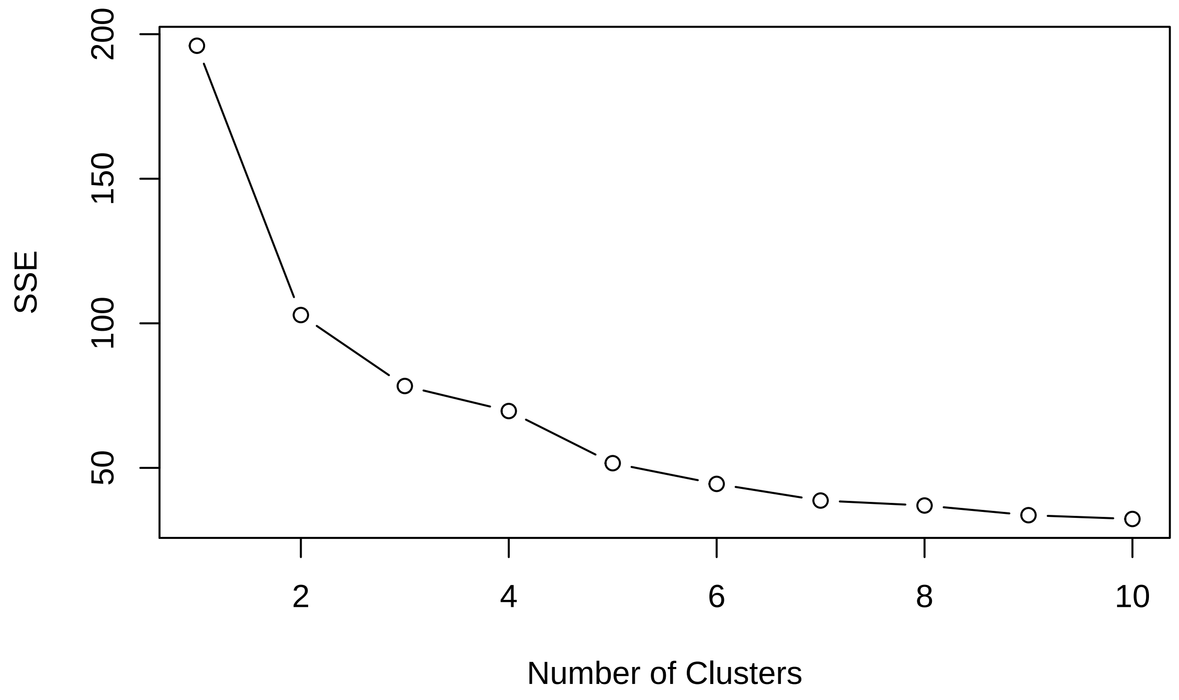
The goal of k-means clustering is to define clusters such that the total intra-cluster variation (known as total within-cluster variation or total within-cluster sum of square) is minimized.

The total within-cluster sum of square (wss) measures the compactness of the clustering and we want it to be as small as possible. Thus, we can use the following algorithm to define the optimal clusters:

1. Compute clustering algorithm (e.g., k-means clustering) for different values of k. For instance, by varying k from 1 to 10 clusters
2. For each k, calculate the total within-cluster sum of square (wss)
3. Plot the curve of wss according to the number of clusters k.
4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

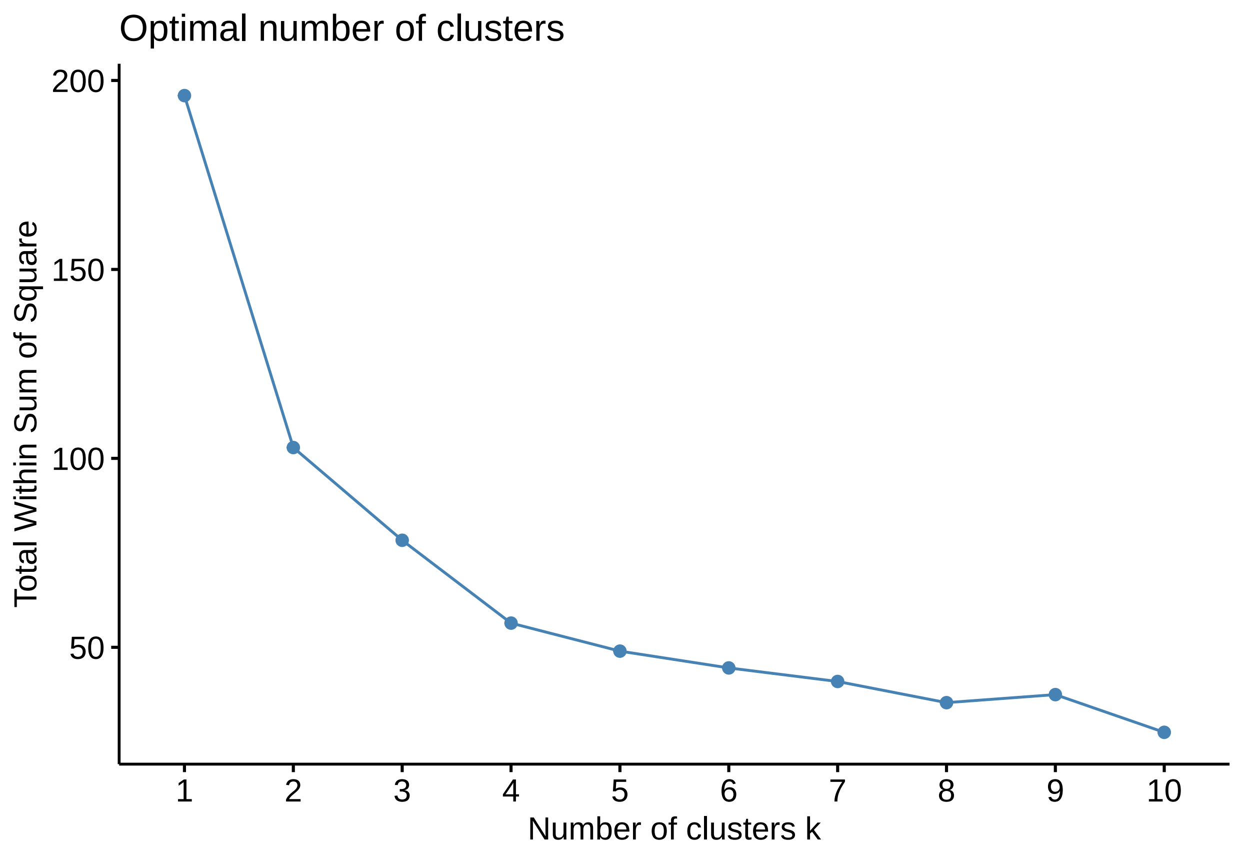
Now, let's try finding the "most appropriate" cluster number using the elbow method. Directly from the output of kmeans(), we can obtain the SSE for a specific clustering outcome.

```r
SSE_curve <- c()
for (n in 1:10) {
  kcluster <- kmeans(df, n)
  #print(kcluster$withinss)
  sse <- sum(kcluster$withinss)
  SSE_curve[n] <- sse
}
plot(1:10, SSE_curve, type="b", xlab="Number of Clusters", ylab="SSE")
```



This process to compute the "Elbow method" has been wrapped up in a single function (fviz_nbclust):

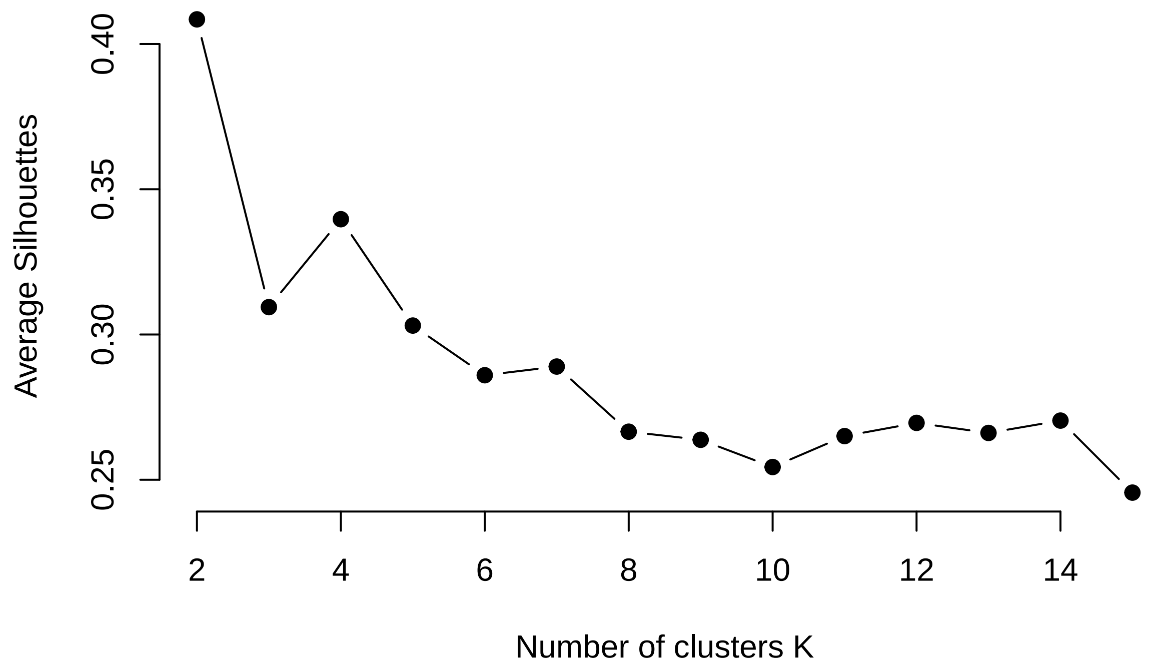
```r
set.seed(1)

fviz_nbclust(df, kmeans, method = "wss")
```



The results suggest that 4 is the optimal number of clusters as it appears to be the bend in the knee (or elbow).

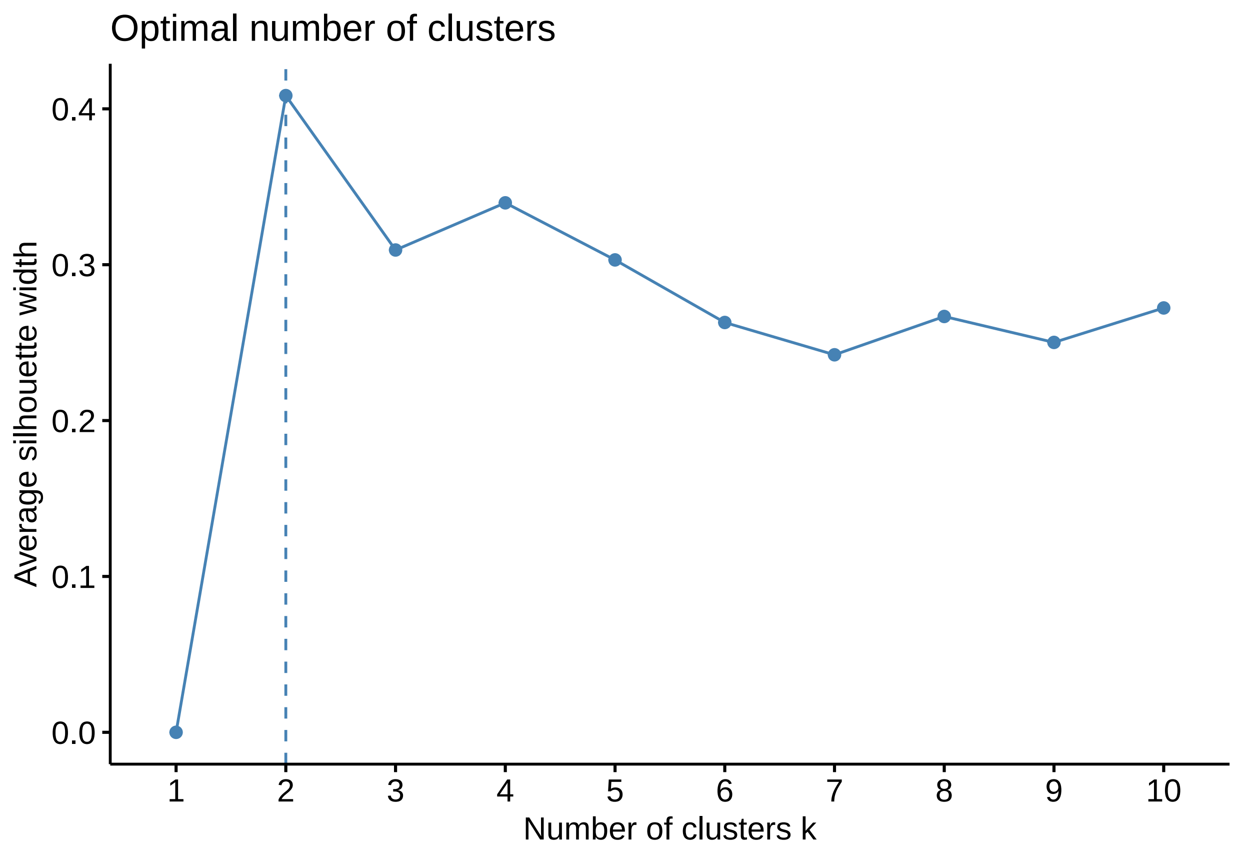
Silhouette method

```r
# function to compute average silhouette for k clusters
avg_sil <- function(k) {
  km.res <- kmeans(df, centers = k, nstart = 25)
  ss <- silhouette(km.res$cluster, dist(df))
  mean(ss[, 3])
}

# Compute and plot wss for k = 2 to k = 15
k.values <- 2:15

# extract avg silhouette for 2-15 clusters
avg_sil_values <- map_dbl(k.values, avg_sil)

plot(k.values, avg_sil_values,
       type = "b", pch = 19, frame = FALSE,
       xlab = "Number of clusters K",
       ylab = "Average Silhouettes")
```



Similar to the elbow method, this process to compute the "average silhoutte method" has been wrapped up in a single function (fviz_nbclust):

```
fviz_nbclust(df, kmeans, method = "silhouette")
```



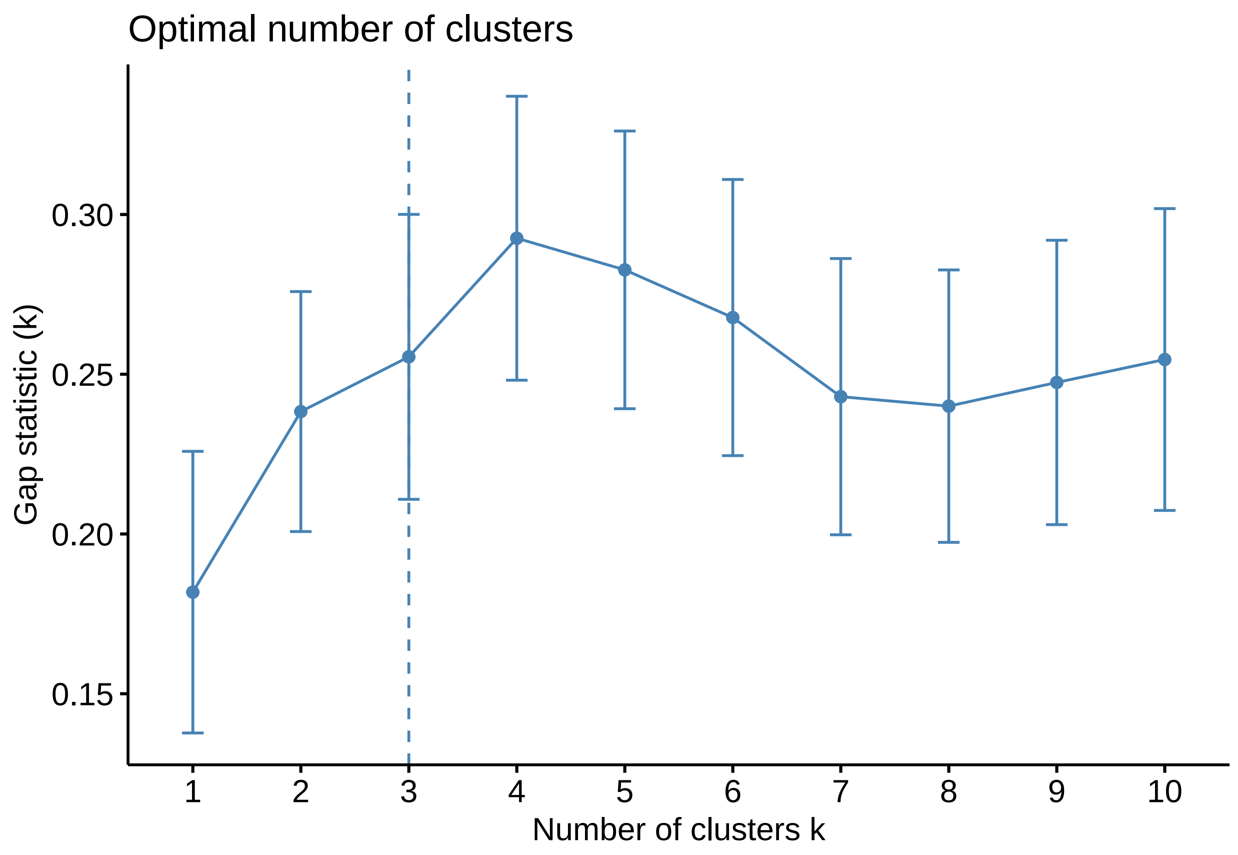
Gap Statistic

The gap statistic has been published by R. Tibshirani, G. Walther, and T. Hastie (Standford University, 2001). The approach can be applied to any clustering method (i.e. K-means clustering, hierarchical clustering). The gap statistic compares the total intracluster variation for different values of k with their expected values under null reference distribution of the data (i.e. a distribution with no obvious clustering). The reference dataset is generated using Monte Carlo simulations of the sampling process.

```r
# compute gap statistic
set.seed(1)
gap_stat <- clusGap(df, FUN = kmeans, nstart = 25,
                    K.max = 10, B = 50)
# Print the result
print(gap_stat, method = "firstmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = df, FUNcluster = kmeans, K.max = 10, B = 50, nstart = 25)
## B=50 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstmax'): 4
##           logW   E.logW       gap     SE.sim
##  [1,] 3.458369 3.640162 0.1817929 0.04407604
##  [2,] 3.135112 3.373427 0.2383156 0.03754968
##  [3,] 2.977727 3.233173 0.2554469 0.04459332
##  [4,] 2.826221 3.118798 0.2925776 0.04443439
##  [5,] 2.738868 3.021538 0.2826697 0.04346192
##  [6,] 2.666967 2.934708 0.2677410 0.04322156
##  [7,] 2.612957 2.855940 0.2429830 0.04323594
##  [8,] 2.545027 2.785050 0.2400225 0.04263093
##  [9,] 2.468162 2.715598 0.2474358 0.04451238
## [10,] 2.394884 2.649495 0.2546114 0.04723832
```

```
fviz_gap_stat(gap_stat)
```

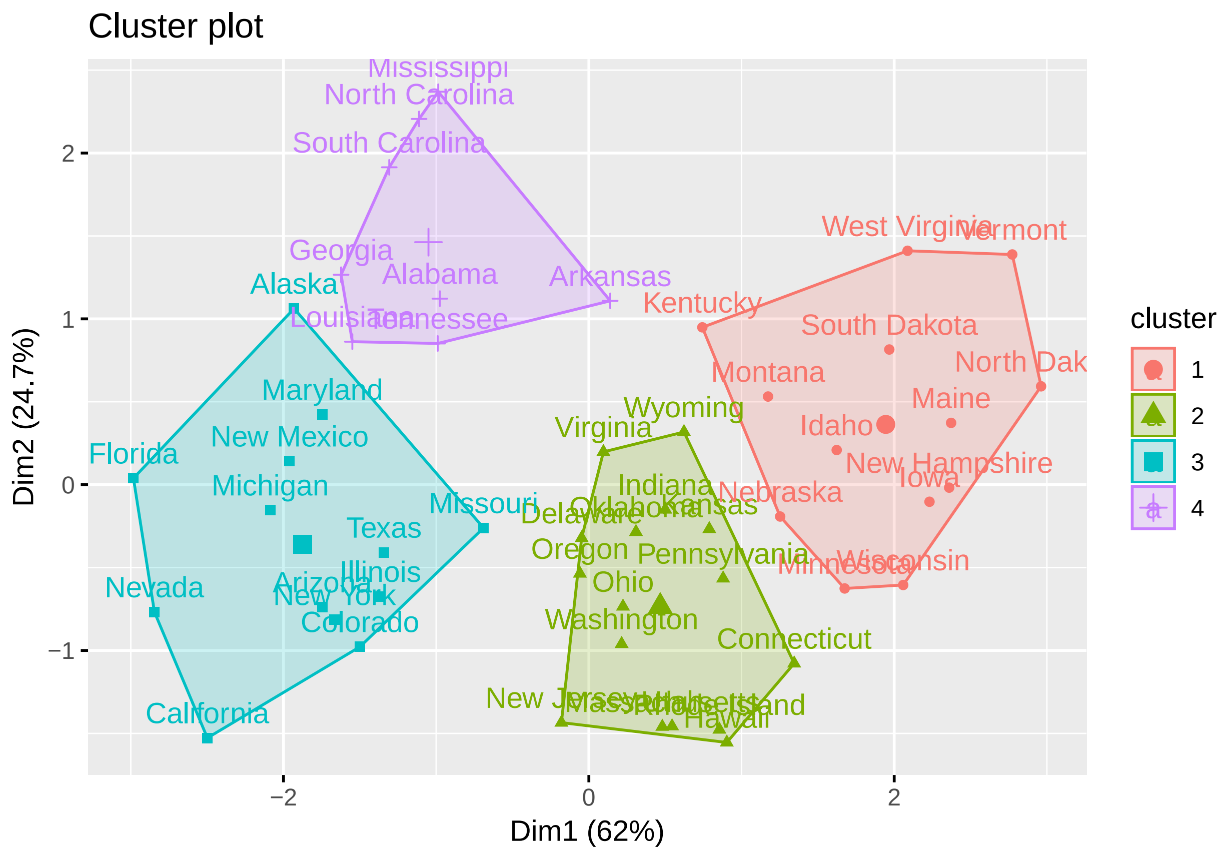


Based on the above results we can perform the final analysis and extract the results using 4 clusters.

```
# Compute k-means clustering with k = 4
set.seed(123)
final <- kmeans(df, 4, nstart = 25)
print(final)
```

```
## K-means clustering with 4 clusters of sizes 13, 16, 13, 8
##
## Cluster means:
##       Murder    Assault   UrbanPop        Rape
## 1 -0.9615407 -1.1066010 -0.9301069 -0.96676331
## 2 -0.4894375 -0.3826001  0.5758298 -0.26165379
## 3  0.6950701  1.0394414  0.7226370  1.27693964
## 4  1.4118898  0.8743346 -0.8145211  0.01927104
##
## Clustering vector:
##        Alabama         Alaska        Arizona       Arkansas     California
##              4              3              3              4              3
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              3              2              2              3              4
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              2              1              3              2              1
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              2              1              4              1              3
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              3              1              4              3
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              1              1              3              1              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              3              3              4              1              2
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              2              2              4
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              1              4              3              2              1
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              2              1              1              2
##
## Within cluster sum of squares by cluster:
## [1] 11.952463 16.212213 19.922437  8.316061
##  (between_SS / total_SS =  71.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
fviz_cluster(final, data = df)
```



Disadvantages

One potential disadvantage of K-means clustering is that it requires us to pre-specify the number of clusters. An additional disadvantage of K-means is that it's sensitive to outliers and different results can occur if you change the ordering of your data. The Partitioning Around Medoids (PAM) clustering approach is less sensititive to outliers and provides a robust alternative to k-means to deal with these situations.