Cluster Analysis Lecture

Luke T. Daniels

4/1/2018

Cluster analysis in R is a great tool to find trends and patterns in data. Many people think of statistics as giving definitive answers to questions, however, there are techniques that simply provide further insight into data. Clustering allows users to identify which observations are alike across many different variables. The power of cluster analysis allows us to perform complex analyses that would be near impossible without programs such as R.

This demonstration serves as an a brief introduction to the statistics behind cluster analysis and the corresponding tools in R. There are many ways to go about cluster analysis. I will focus on the *Partitioning Method* and the *Heirichal Method using Agglomerative Clustering*. This demonstration will also cover ways that we create solid analyses. It will cover the the *Hopkins Statistic* which tells whether our data is actually clusterable. Furthermore, the demonstration will show how to obtain p-values after our data has been clustered! I will highlight the differences in base `R` and packages such as `factoextra` and `cluster`.

As mentioned, there are many ways to go about cluster analysis. This demonstration will cover the most popular. However, it will not go in depth with the statistical details. I highly recomend researching the statistics if you are interested because it is quite amazing! Lastly, I would like to thank Alboukadel Kassambara for writing the Book 'Practical Guide to Cluster Analysis in R.' This demonstration is largely based on his work and I must give credit to where it is due.

Road Map

- **Determining If Clusters Exist**
- **How To Calulate Multivariate Distance**
- **K-Means Clustering**
- **PAM Clustering**
- **Hierarichal Clustering Using Agglomerative Testing**
- **Graphical Manipulation**
- **Obtaining P-Values**

How Does A Cluster Analysis Deal With Multivariate Data?

A cluster analysis divides data into groups. The groups, or clusters, should capture the natural structure of the data. Cluster Analysis is synonomous with multivariate analysis. In other words, clusters are being assigned based on multiple dimensions. How is this possible?

1.) The first step to cluster analysis is to make sure our data actually contain clusters! A big problem in this field, is that people coerce their data into clusters when clusters do not exist!

2.) If we determine that there are clusters the process begins with the calculation of multivariate distances. These distances will be represented in a n x n matrixs of D. This calculation allows us to compare one element of two categories, with another element of two different categories. We must scale the data set for this step!

3.) Next we must determine the clustering method (Heirarchical or Partitioning). You will soon see the differences between these. At this stage we must also carefully think about our data. Does it have categorical variables? Does it have large outliers? These will be important considerations in deciding the test.

4.) Run the cluster analysis and separate data into groups!

5.) Validate the Results and Obtain P- Values!

Required Packages and Data Preparation

* Rows must be observations and columns must be variables. * Missing values must be removed or estimated. * The data must be standardized to make variables comparable. * The data used in cluster analysis can be interval, ordinal or categorical. Purely Numeric Data is prefered.

```
# Required Packages
library(cluster)
library(factoextra)
```

```
## Loading required package: ggplot2
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```
library(ggplot2)
library(NbClust)
```

```
# First Data Set to Use - The Built In 'swiss'
df <- swiss
df <- na.omit(swiss) #Remove any missing values that are present
df.scaled <- scale(df) # Scale the data
head(df, n =3)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
##              Infant.Mortality
## Courtelary               22.2
## Delemont                 22.2
## Franches-Mnt             20.2
```

Why Do We Need to Scale?

One very important decision that needs to be made involves the scales of the variables being measured. If one of the variables is measured on a much larger scale than the other variables, then whatever measure is

used will be overly influenced by that variable. For example, if we are looking at the distance between two people based on their IQs and incomes in dollars, the differences in incomes would dominate the distance measurments. We can solve this issue by standardizing the variables! bo

Should I Include Every Variable?

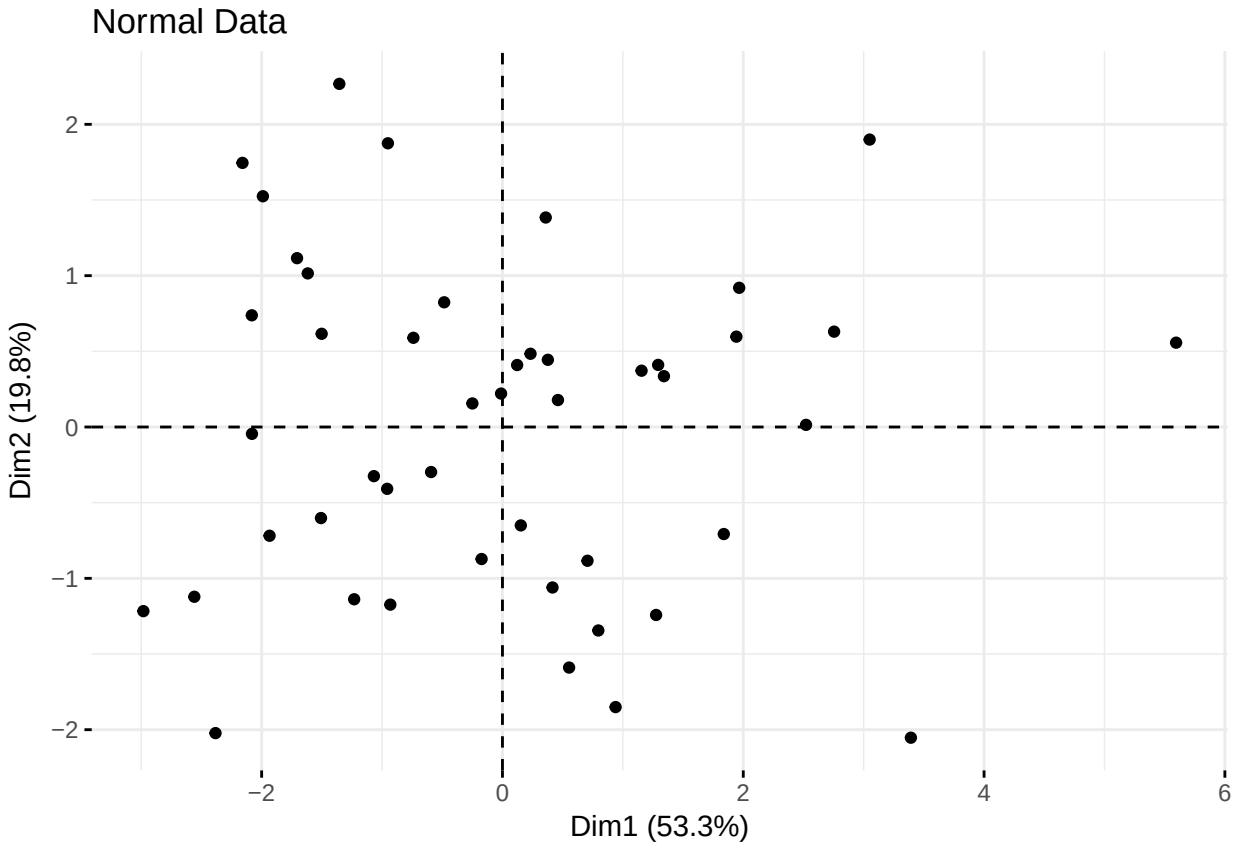
Cluster analysis has no mechanism for differentiating between relevant and irrelevant variables.There must be careful consideration for the variables included in the analysis. There should be signficant differences between the variables. There are select pricipals that should be followed.

- Avoid using an abundance of variables since this increases the odds that variables are no longer dissimilar.
- If there is a high degree of collinearity, specific aspects covered by these variables will be overrepresented.
- Formann (1984) recommends a sample size of at least 2^m where m equals the number of clustering variables

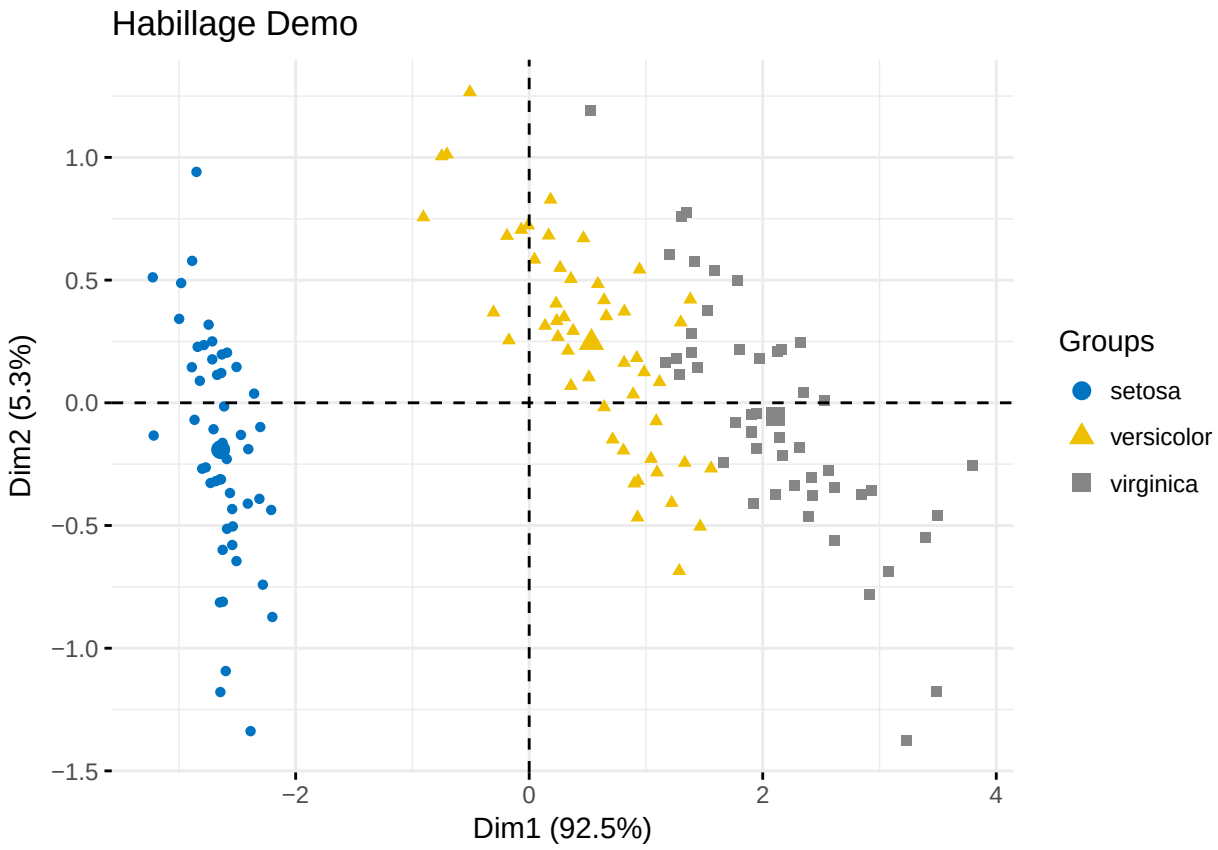
Step 1: Do Clusters Exist In Our Data

Cluster Analysis will return clusters even if the data does not contain any clusters. This can seriously lead us astray in our interpretation of the data! Thankfully there are statistical methods to determine is clusters exist! A helpful method is to randomize your data set, so you can compare the observed and randomized. For this step we will use the `fviz_pca_ind()` function from the `factoextra` package. So see the inputs for this function please see the appendix.

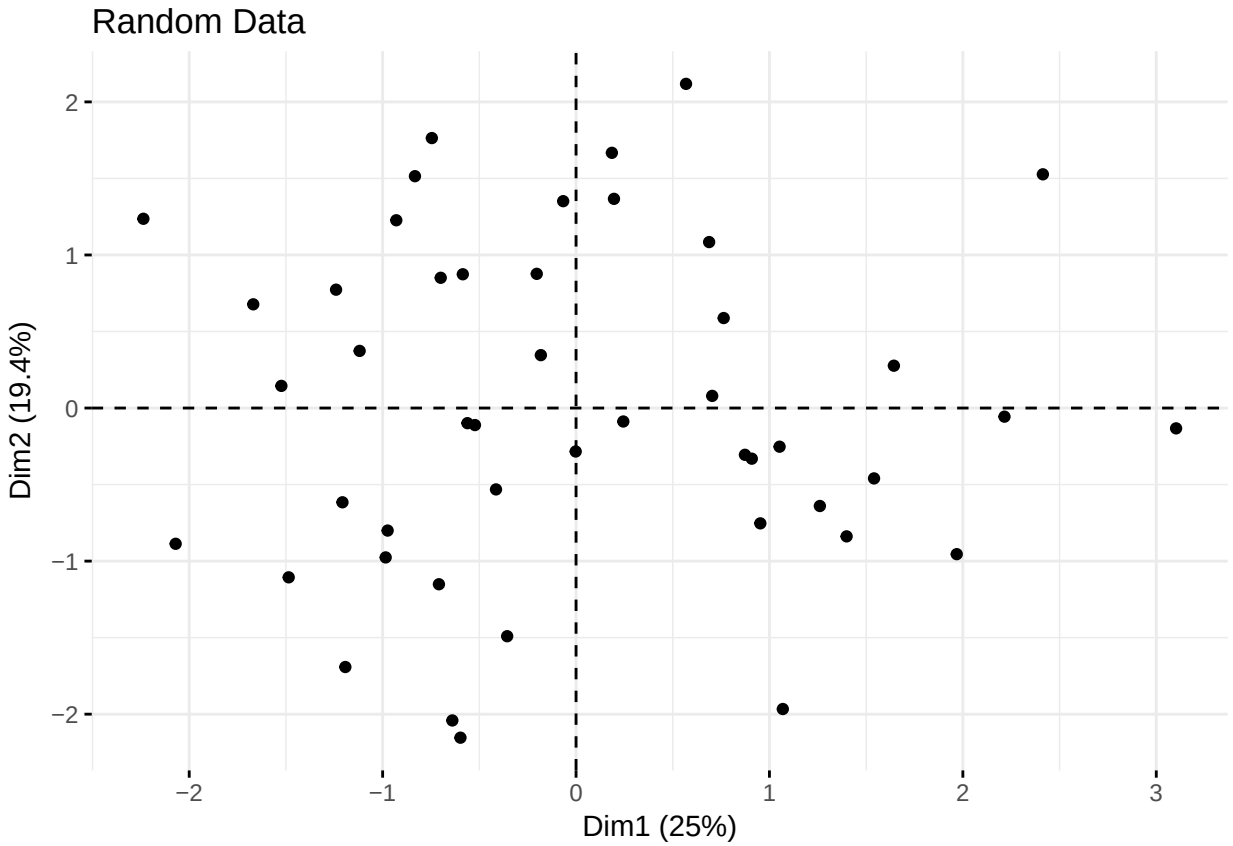
```
# Randomize the Swiss Data
set.seed(123)
random_df <- apply(df, 2, function(x){runif(length(x),
                                            min(x), (max(x)))})
random_df <- as.data.frame(random_df)

# Standardize the Data

swiss.scaled <- scale(df)
random_df.scaled <- scale(random_df)

# We can view the two data sets using factoextra and a Principle COmponent Analysis
# The fviz_pca is a pricipal component analysis that reduces the dimensionality of the multivariate dat

fviz_pca_ind(prcomp(swiss.scaled), title = "Normal Data",
             palette = "jco", geom = "point")
```




```
# The habillage function is useful when there is a categorical variable in the data

fviz_pca_ind(prcomp(iris[,-5]), title = "Habillage Demo", habillage = iris$Species,
             palette = "jco", geom = "point")
```




```
# Back to the Swiss Data

fviz_pca_ind(prcomp(random_df.scaled), title = "Random Data", geom = "point")
```




```
# By comparing the two graphs we can see that the observed data is different from the randomized data

# The Hopkins test, tests the spatial randomness of the data
# If the Hopkins Stat is <0.5 then it is unlikely that the data has significant clusters
# We use get_clust tendency in the factoextra package

Hopkins <- get_clust_tendency(swiss.scaled, n = nrow(swiss.scaled)-1, graph = TRUE)
Hopkins$hopkins_stat  # To get the Stat
```

```
## [1] 0.3089652
```

```
# The Stat is 1 - 0.308.  Our Hopkins Stat is 0.692!

RandomHopkin <- get_clust_tendency(random_df.scaled, n = nrow(swiss.scaled) - 1, graph = TRUE)
RandomHopkin$hopkins_stat
```

```
## [1] 0.4885209
```

```
# Our Hopkin Stat is 0.52!

# With a Hopkin Stat of 0.692 we can determine that the swiss data set has clusters.  Furthermore
# it has a higher stat than the random data set.
```

The Hopkins Test determined that the swiss data set has clusters in it. To this point, we do not know how many clusters exist or where. The Hopkins Statistic (Lawson and Jurs 1990) measures the probability that a given data set is generated by a uniform data distribution. (i.e - spatial randomness). The `get_clust_tendency` assesses clustering tendency using Hopkins' statistic.

You may have noticed the function `fviz_pca_ind`. This function conducts a principle component analysis. The algorithm behind this wrapper dives into linear algebra so I will explain it at a high level. Priciple Components allows us to summarize and visualize the information in a data set with many variables. It shinks a data set with many variables down to two variables, called principle components. These two variables correspond to a linear combination of the original data. It also represents the total variation that the component contains (Dim1 and Dim2).

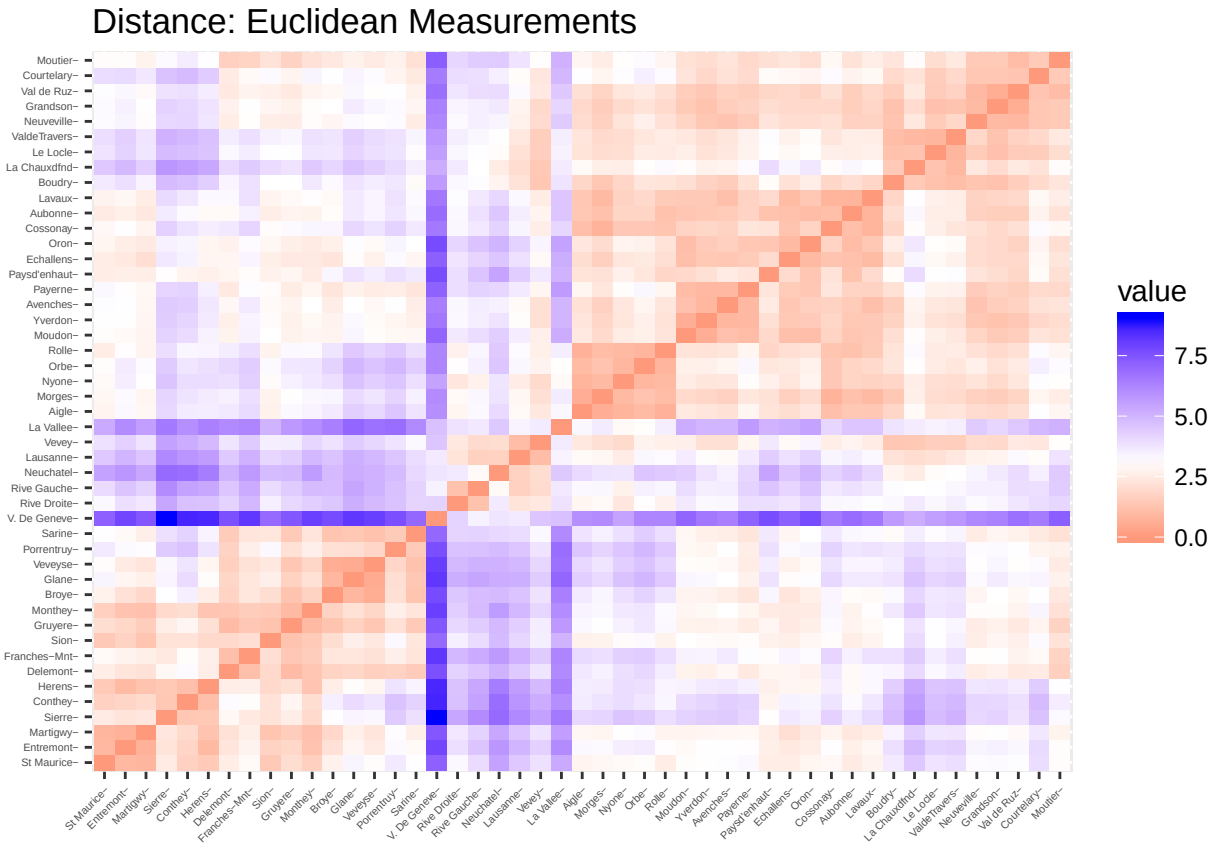
Step 2: Calculating Multivariate Distances

There are may R functions for computing distances between pairs of observations

- dist() - Base R - Accepts only numeric data
- get_dist() - factoextra package - Accepts only numeric data, but supports correlation based distance measures!
- daisy() - cluster package - Able to handle any type of variable

Within each function, there are different ways to calculate the distance. These include the most popular `Euclidian`, but also `Manhattan`, `Pearson`, `Spearman`, and `Kendall`. Each method has advantages. For example Manhattan is better for outliers, and Pearson approaches the measurements but also taking into account correlation. I will focus on `Euclidian` for this demo, but I encourage you to research the other methods to make you analysis more powerful!

```
disteucl <- get_dist(x = df, method = "euclidean", stand = TRUE) #Stand = TRUE indicates that the varia
euclmatrix <- as.matrix(disteucl)

## But what if we Have variables that are binary or categorical? Use Daisy with the "gower" distance.

distgower <- daisy(df.scaled, metric = "gower", stand = FALSE) #with stand = FALSE we must provide the

gowermatrix <- as.matrix(distgower)

# Visualizing the Distance Matrix

DistanceMap <- fviz_dist(disteucl, order = TRUE, show_labels = TRUE, lab_size = 4) + labs(title = "Dista
# You can change the color gradient using gradient = list(low = "red", mid = "white", high = "blue")


DistanceMap #You may have noticed that this is similar to a ggplot.  The factoextra package is highly i
```



Partitioning Clustering: K-Means and K-Mediod

The first of our clustering methods is Partitioning clustering. This is a method to classify observation into groups based on similarity. The main difference between Partitioning Vs Hierarchical is that in the former the user has to specify the number of clusters.

Part I: K- Means: Paritioning Cluster

How does the Algorithm Work?

Basically K- mean is an interative process that divides a given data set into K disjoint groups

It starts by placing k centroids randomly in your space. Then you iteraviley do the following: First we run through our data set and for each individual we find the nearest centroid to that individual. To do that, for each x_i you compute the distance between in a c_j (each cluster). This is the Eucleadian Distance. Then you pick the cluster that has the minimum distance of all (nearest cluster).

Then you assign x_i to that nearest cluster. This process occurs for each individual, so each individual minimizes is distances to the randomly positioned centroid. Now we need to recompute the centroid by getting the average of all the x_i that was assigned to that cluster. (All the x_i's that were assigned to the jth cluster and you average them out.)

At this point we are restricting the analysis to continous variables. We cannot take the average or distance of categorical variables.

These are the two basic steps. You keep running theses steps until no individuals change cluster memberships

```
# Using Base R for Calculating Cluster Means
set.seed(123) #since this algorithm starts with k randomly selected centroids, its recomended to set th
km.res <- kmeans(df.scaled, centers = 3, iter.max = 250, nstart =25) #nstart is recomended to be 25 - 5
head(km.res)
```

```
## $cluster
##   Courtelary     Delemont Franches-Mnt      Moutier   Neuveville
##            1            2            2            1            1
##   Porrentruy        Broye        Glane      Gruyere       Sarine
##            2            2            2            2            2
##      Veveyse        Aigle      Aubonne     Avenches     Cossonay
##            2            1            1            1            1
##    Echallens     Grandson     Lausanne    La Vallee       Lavaux
##            1            1            3            3            1
##       Morges       Moudon        Nyone         Orbe         Oron
##            1            1            1            1            1
##      Payerne Paysd'enhaut        Rolle        Vevey      Yverdon
##            1            1            1            3            1
##      Conthey    Entremont       Herens     Martigwy      Monthey
##            2            2            2            2            2
##   St Maurice       Sierre         Sion       Boudry La Chauxdfnd
##            2            2            2            1            3
##     Le Locle    Neuchatel   Val de Ruz ValdeTravers V. De Geneve
##            1            3            1            1            3
##  Rive Droite  Rive Gauche
##            3            3
##
## $centers
##     Fertility Agriculture Examination  Education   Catholic
## 1 -0.09146501  0.01020332   0.1294049 -0.2871779 -0.7980284
## 2  0.83314915  0.65426591  -0.8839264 -0.4527862  1.3189394
## 3 -1.40333639 -1.33786636   1.3958136  1.7312087 -0.3435471
##   Infant.Mortality
## 1      -0.06984001
## 2       0.28579935
## 3      -0.37080867
##
## $totss
## [1] 276
##
## $withinss
## [1] 45.23153 41.67497 37.29864
##
## $tot.withinss
## [1] 124.2051
##
```

```
## $betweenss
## [1] 151.7949
```

```
# Its possible to add the cluster assignments to the original data set
aggregate(swiss, by=list(cluster = km.res$cluster), mean)
```

```
##   cluster Fertility Agriculture Examination Education  Catholic
## 1       1   69.0000    50.89130    17.52174  8.217391  7.862174
## 2       2   80.5500    65.51875     9.43750  6.625000 96.150000
## 3       3   52.6125    20.27500    27.62500 27.625000 26.816250
##   Infant.Mortality
## 1         19.73913
## 2         20.77500
## 3         18.86250
```

```
swiss2 <- cbind(swiss, cluster = km.res$cluster)
head(swiss2)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality cluster
## Courtelary               22.2       1
## Delemont                 22.2       2
## Franches-Mnt             20.2       2
## Moutier                  20.3       1
## Neuveville               20.6       1
## Porrentruy               26.6       2
```

```
# We can also access the various pieces of information in kmeans

km.res$size #Items in Each cluster
```

```
## [1] 23 16  8
```

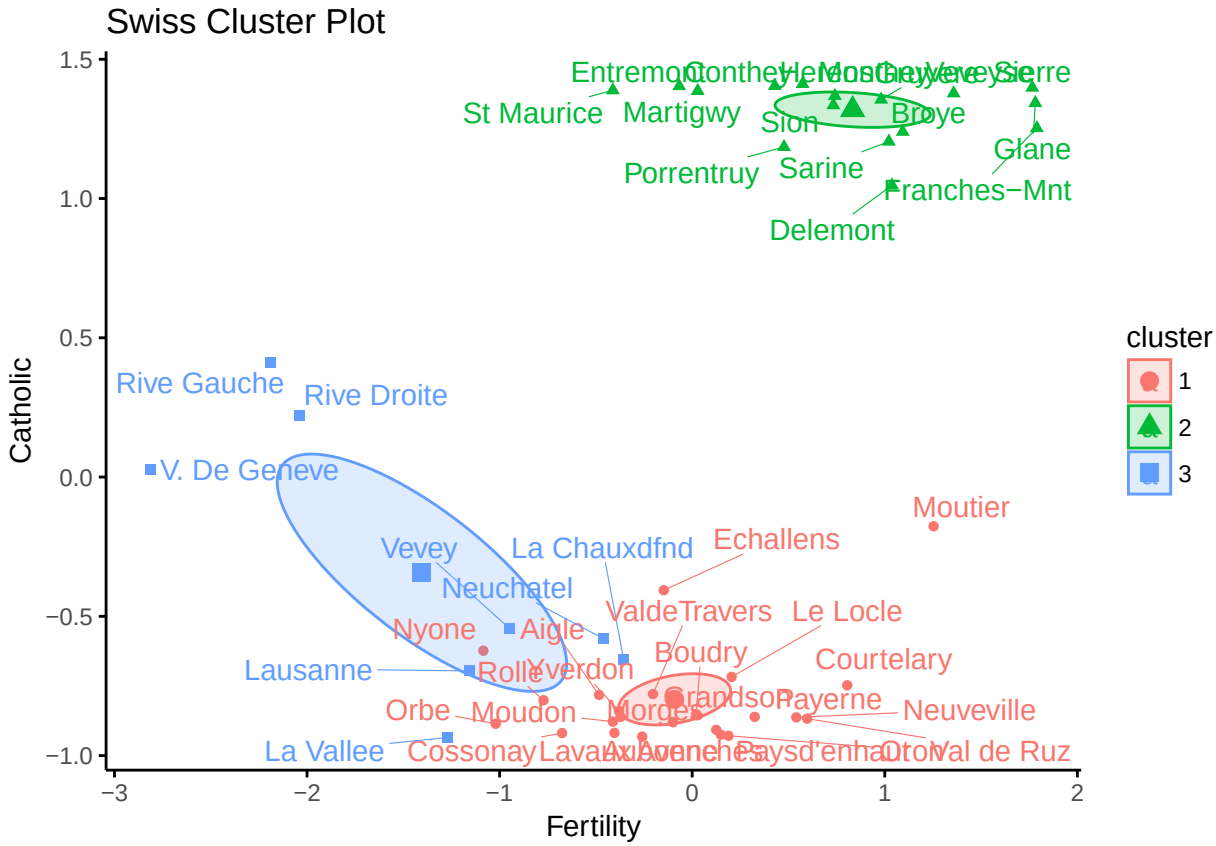
```
km.res$centers # The Cluster Means
```

```
##     Fertility Agriculture Examination  Education   Catholic
## 1 -0.09146501  0.01020332   0.1294049 -0.2871779 -0.7980284
## 2  0.83314915  0.65426591  -0.8839264 -0.4527862  1.3189394
## 3 -1.40333639 -1.33786636   1.3958136  1.7312087 -0.3435471
##   Infant.Mortality
## 1      -0.06984001
## 2       0.28579935
## 3      -0.37080867
```

```
km.res$tot.withinss #Total Within Sum of Squares
```

```
## [1] 124.2051
```

```
# Visualizing K-means
fviz_cluster(km.res, data = df.scaled, choose.vars = c("Fertility", "Catholic"), stand = FALSE, geom = c
```



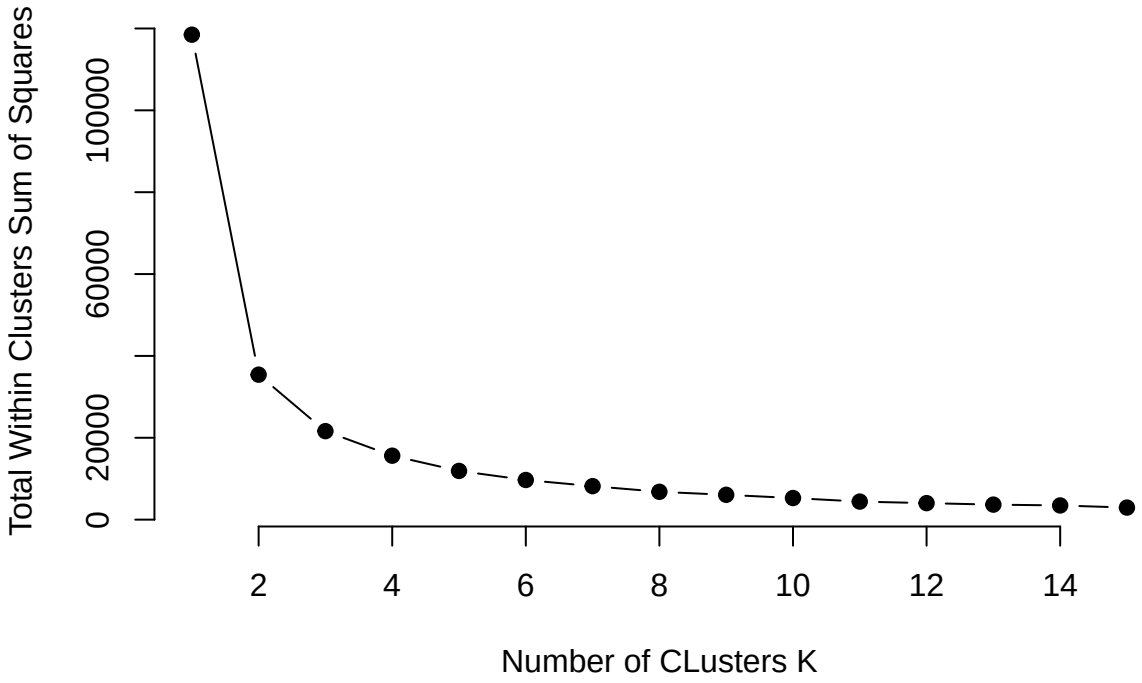
What is the Right Number of Clusters?

You may have questioned how I chose to form 3 clusters. That was a randomly selected number. Now you may start to see the difficulty of k-means clustering. Determining the number of clusters is one of the most important steps in this process. Thankfully, there are many ways to help us determine the number of clusers in the data.

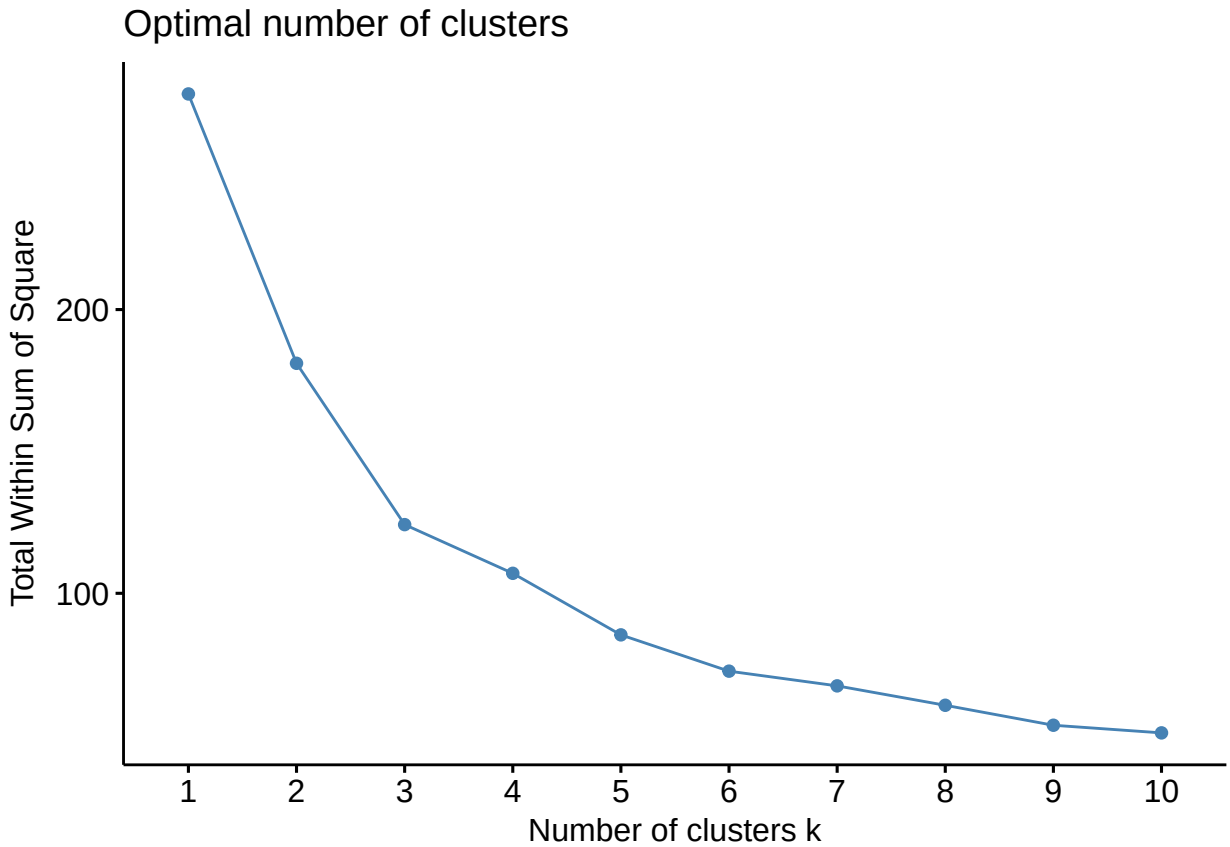
There are three methods for determining the optimal number of clusters: * Method 1: Elbow Method * Method 2: Silhouette Method * Method 3: Gap Statistic

Method 1: Elbow Method

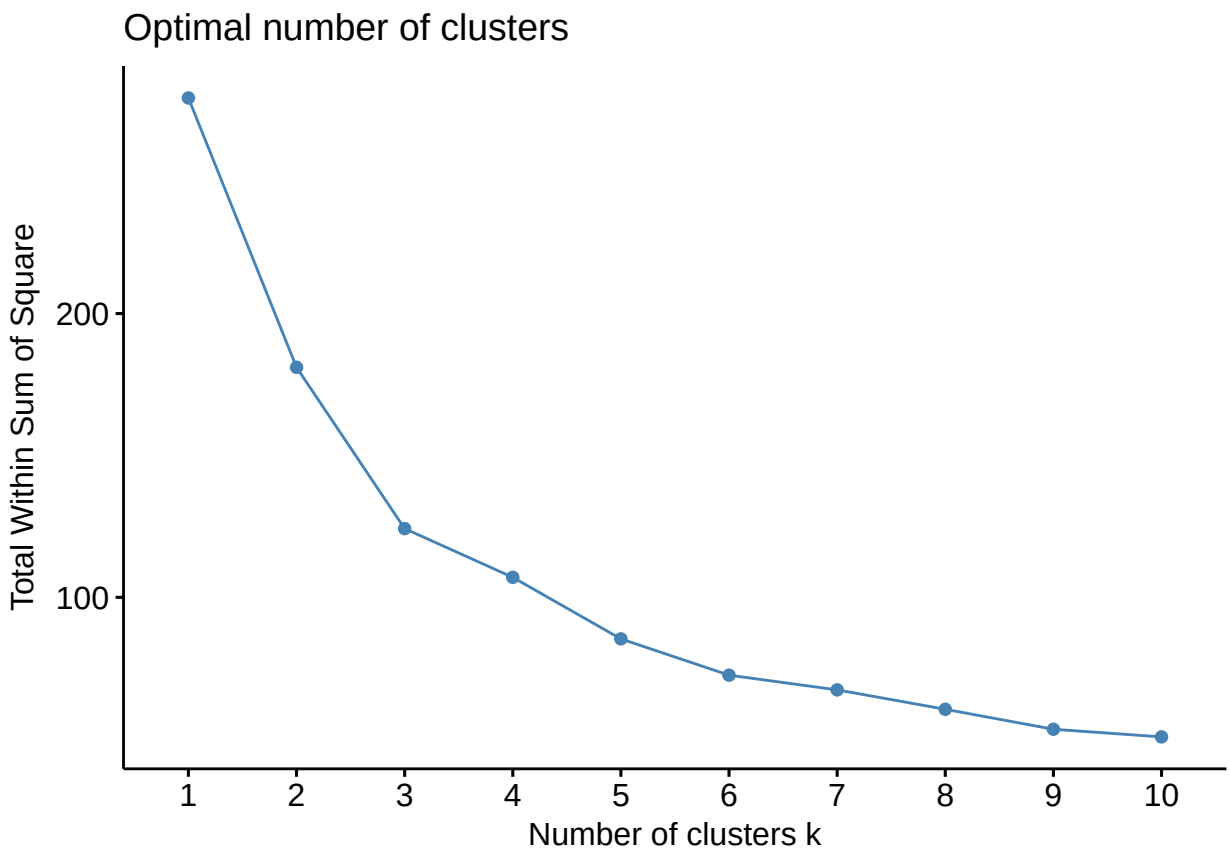
```
### Method 1: Elbow Method In Base R

k.max <- 15 # we want to look at the within squares for clusters 2-15.
wss <- sapply(1:k.max, function(k){kmeans(df, k, nstart = 50)$tot.withinss})
plot(1:k.max, wss, type = "b", pch = 19, frame = FALSE, #type = b indicates points joined by lines
     xlab = "Number of CLusters K",  #pch is the shape of the point, 19 = circle
     ylab = "Total Within Clusters Sum of Squares")
```




```
### Method 1: Elbow Method Using FactoExtra
fviz_nbclust(df.scaled, FUNcluster = kmeans, method = "wss")
```



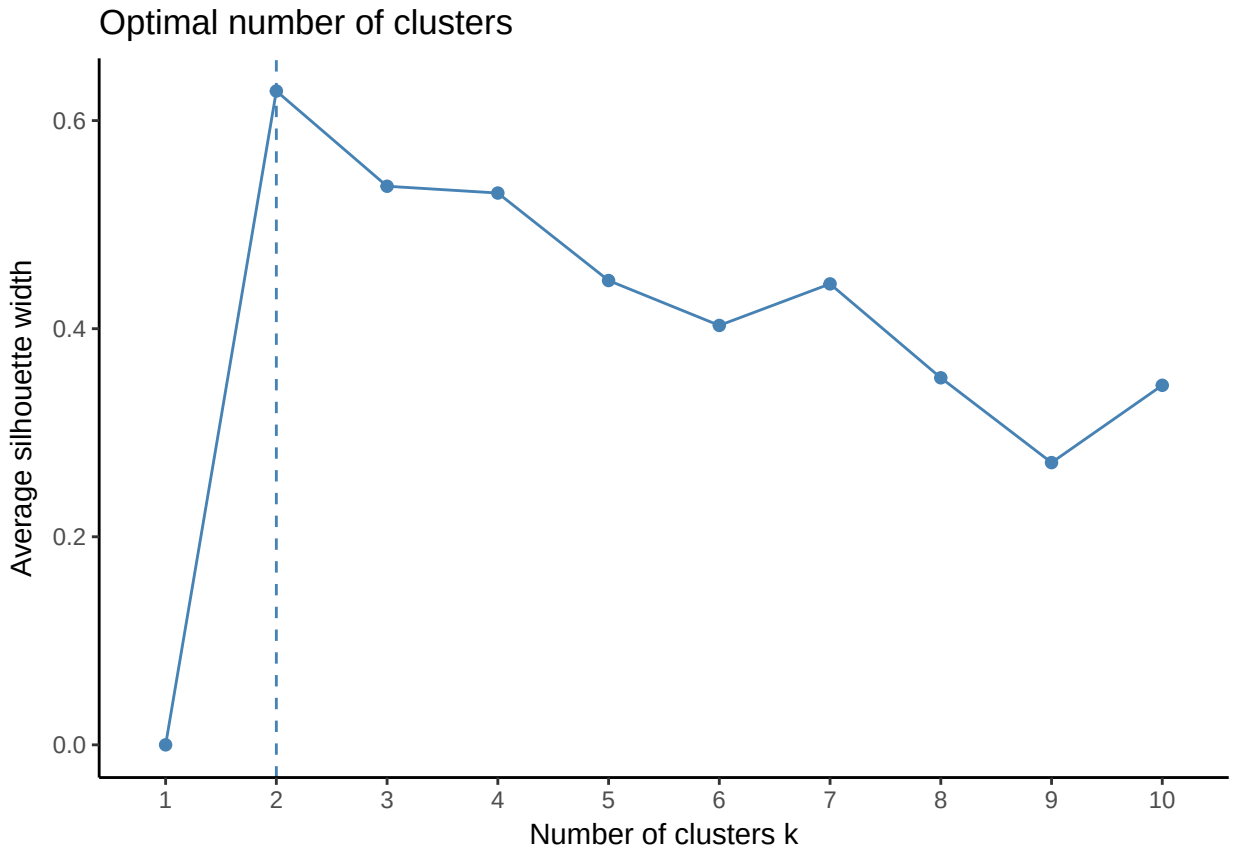
```r
fviz_nbclust(df.scaled, FUNcluster = kmeans, method = "wss")
```



Method 2: Silhouette Method

```r
### Method 2: Silhouette in Base R

avg_sil <- function(k){
  km.res <- kmeans(df.scaled, 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

#extracting silhouettes
#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")
```

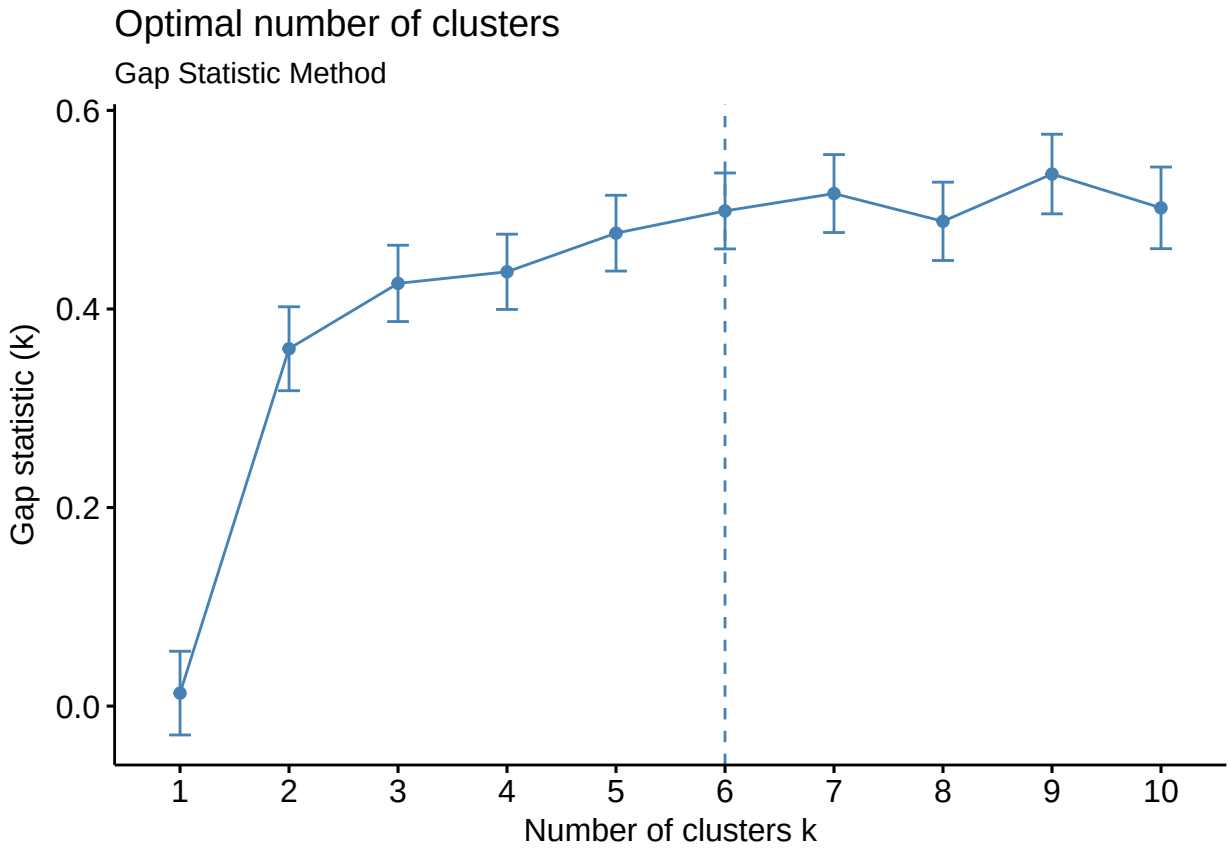
```
### Method 2: Silhouette in FactoExtra
fviz_nbclust(df, kmeans, method = "silhouette") + theme_classic()
```



Method 3: GAP Statistic

The gap statistic is fairly new (R. Tibshirani, G. Walther, and T.Hastie (Standford University, 2001.). The underlying process of the Gap statistic is incredibly complicated, so I will not discuss it too much. Essentially the Gap Statistic compares the total within intra cluster variation for different values of k with their expected values. The optimal clusters will be the smallest values of k such that the gap is within one standard deviation of the gap at k + 1.

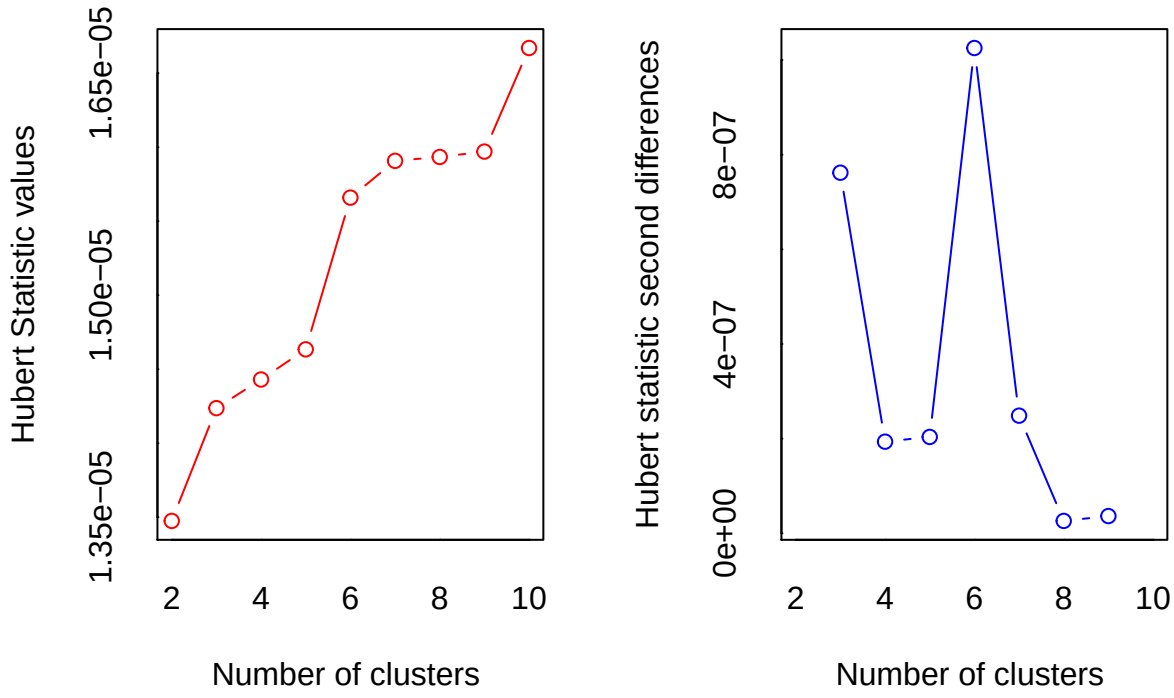
```
### GAP STAT Using Facto Extra

fviz_nbclust(df, kmeans, nstart = 25, method = "gap_stat", nboot = 500) + #nboot is # of bootstrap samp
labs(subtitle = "Gap Statistic Method")
```



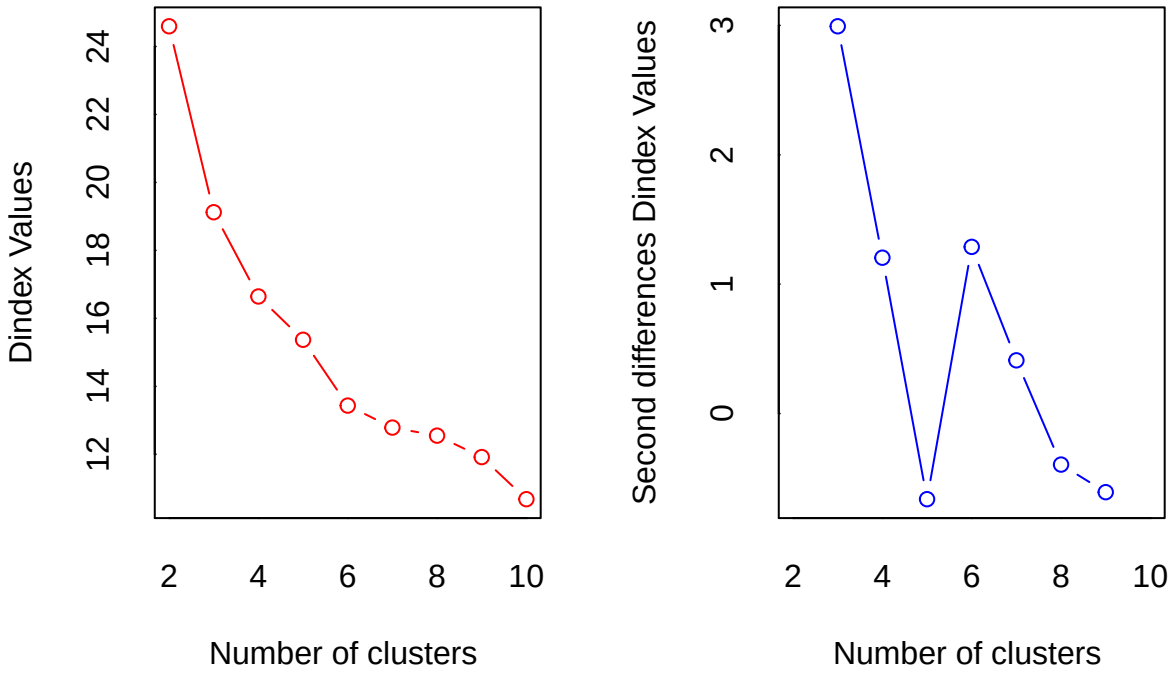
Or We Can Use One Line of Code to Test 30 Different Methods at Once

NbClust Package privdes 30 indeces for determining the number of clusters, distance measures, and clustering methods.

```
nb <- NbClust(df, distance = "euclidean", min.nc = 2,max.nc = 10, method = "kmeans")
```




```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```

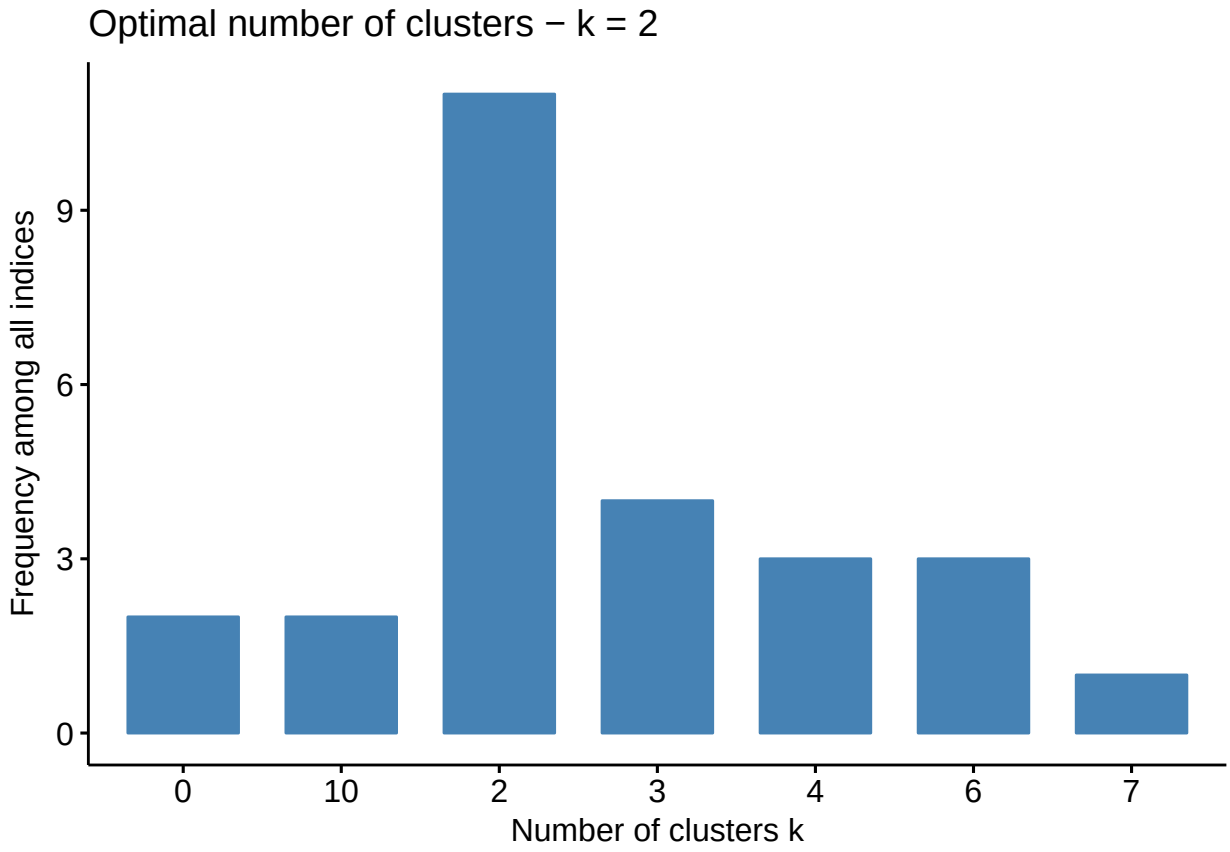



```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
```

```
##
## *******************************************************************
## * Among all indices:
## * 11 proposed 2 as the best number of clusters
## * 4 proposed 3 as the best number of clusters
## * 3 proposed 4 as the best number of clusters
## * 3 proposed 6 as the best number of clusters
## * 1 proposed 7 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

```
fviz_nbclust(nb)
```

```
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 11 proposed  2 as the best number of clusters
## * 4 proposed  3 as the best number of clusters
## * 3 proposed  4 as the best number of clusters
## * 3 proposed  6 as the best number of clusters
## * 1 proposed  7 as the best number of clusters
## * 2 proposed  10 as the best number of clusters
##
## Conclusion
## ==========================
## * According to the majority rule, the best number of clusters is  2 .
```



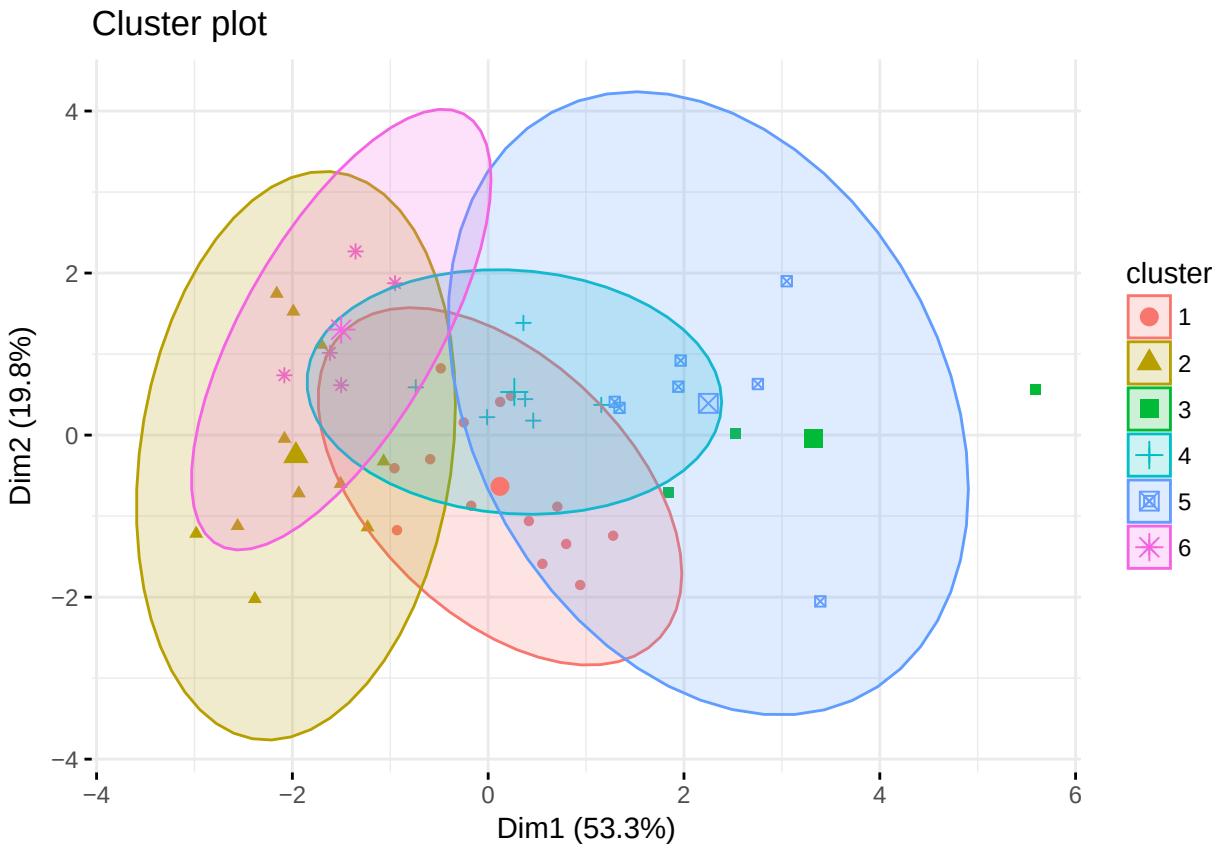
We see condradicting results between the different methods. Some tests suggest 6 clusters while the NbClust suggests 2. To move forward its important that we validate our results.

Validating Our Results

At this point I will introduce a two new functions in the factoextra package. I will focus first on fviz_silhouette(). This is a cluster validation approach that measures how well an observation is clustered and it estimates the average distance between clusters. A negative value means that the the observation is place in the wrong cluster.

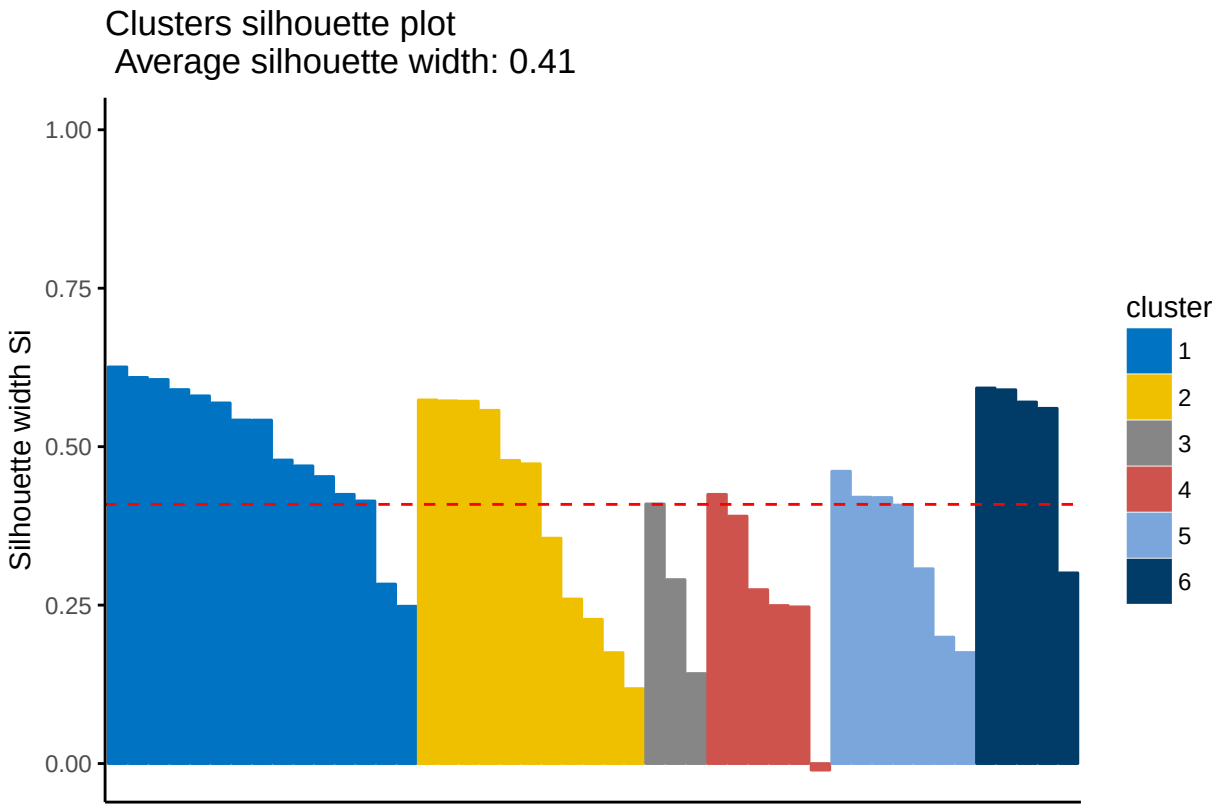
```
km.res <- eclust(df, "kmeans", k = 6, nstart = 25, graph = FALSE)
fviz_cluster(km.res, geom = "point", ellipse.type = "norm", ggtheme = theme_minimal())
```

```
## Too few points to calculate an ellipse
```




```
fviz_silhouette(km.res, palette = "jco", ggtheme = theme_classic())
```

```
##   cluster size ave.sil.width
## 1       1   15          0.50
## 2       2   11          0.40
## 3       3    3          0.28
## 4       4    6          0.26
## 5       5    7          0.34
## 6       6    5          0.52
```



We have seen that some observations have negative values and are thus in the wrong cluster. We are able to observe and extract these observations that are in the wrong cluster.

```
# Extracting The Observations that are in the wrong cluster!
silinfo <- km.res$silinfo


silinfo$clus.avg.widths
```

```
## [1] 0.4957119 0.3965633 0.2805629 0.2625898 0.3415617 0.5227898
```

```
# Determining the bad observations
head(km.res$silinfo$widths) # Look at the width of each observation
```

```
##          cluster neighbor sil_width
## Aubonne        1        4 0.6257486
## Cossonay       1        4 0.6092094
## Lavaux         1        4 0.6059763
## Rolle          1        4 0.5900000
## Aigle          1        4 0.5801217
## Morges         1        4 0.5689809
```

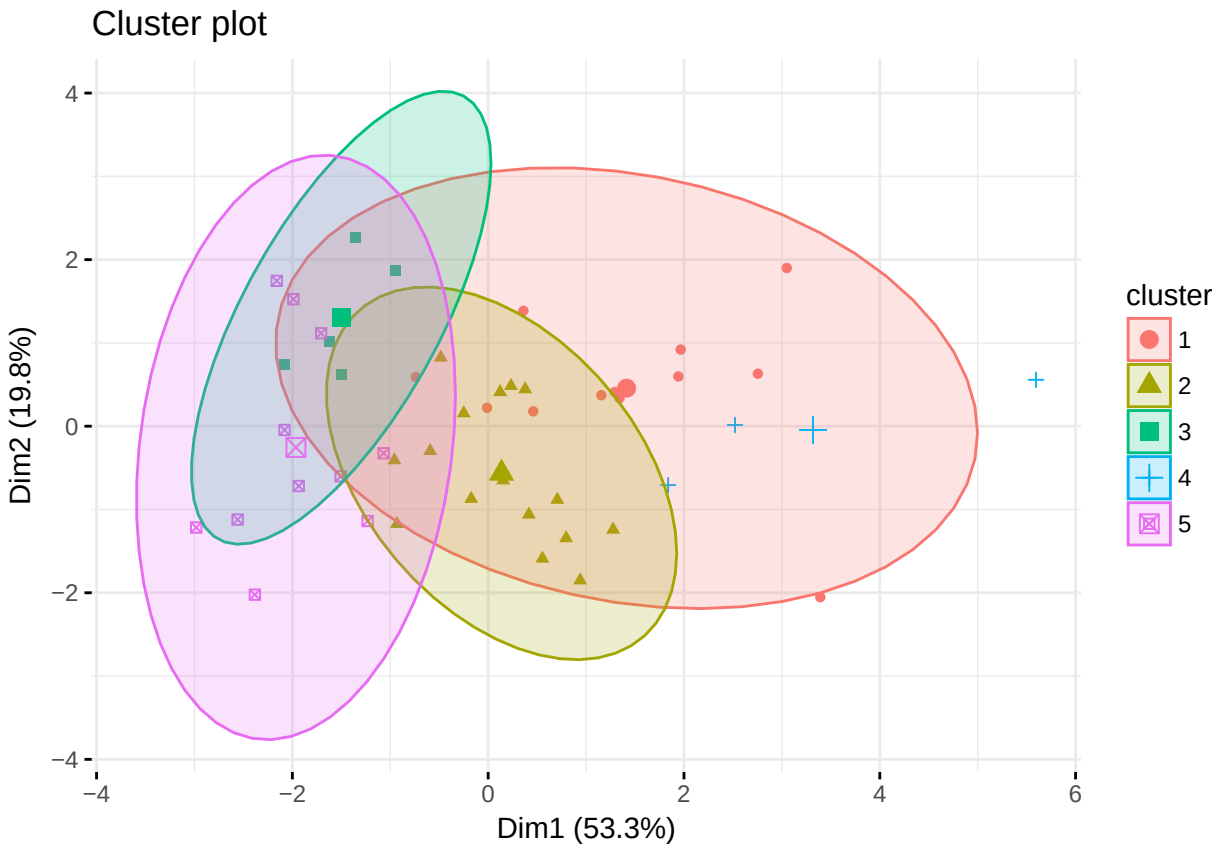
```
sil <- km.res$silinfo$widths

neg_sil_index <- which(sil[, 'sil_width'] < 0)
sil[neg_sil_index, , drop = FALSE] # We see that Courtelary is in the wrong cluster
```

```
##            cluster neighbor   sil_width
## Courtelary       4        5 -0.01032828
```

So what went wrong? We only had one observation that was assigned to the wrong cluster, but that means we may have clustered the data incorectly. Luckily, the function eclust() can calculated the correct correct number of clusters. In the code above, we overrode that ability by saying that there were 6 clusters. If we don't override the function, it will calculate the GAP statistic!

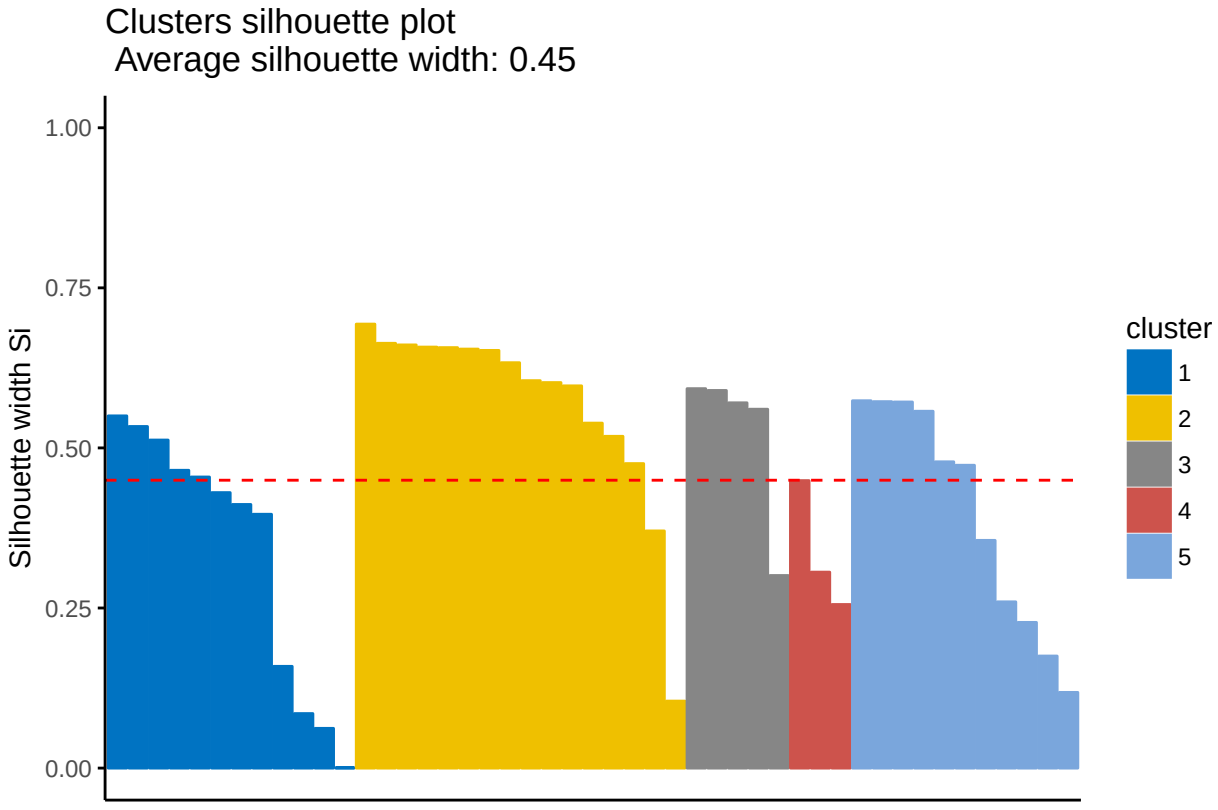
```
km.res <- eclust(df, "kmeans", nstart = 25, graph = FALSE)
fviz_cluster(km.res, geom = "point", ellipse.type = "norm", ggtheme = theme_minimal())
```

```
## Too few points to calculate an ellipse
```



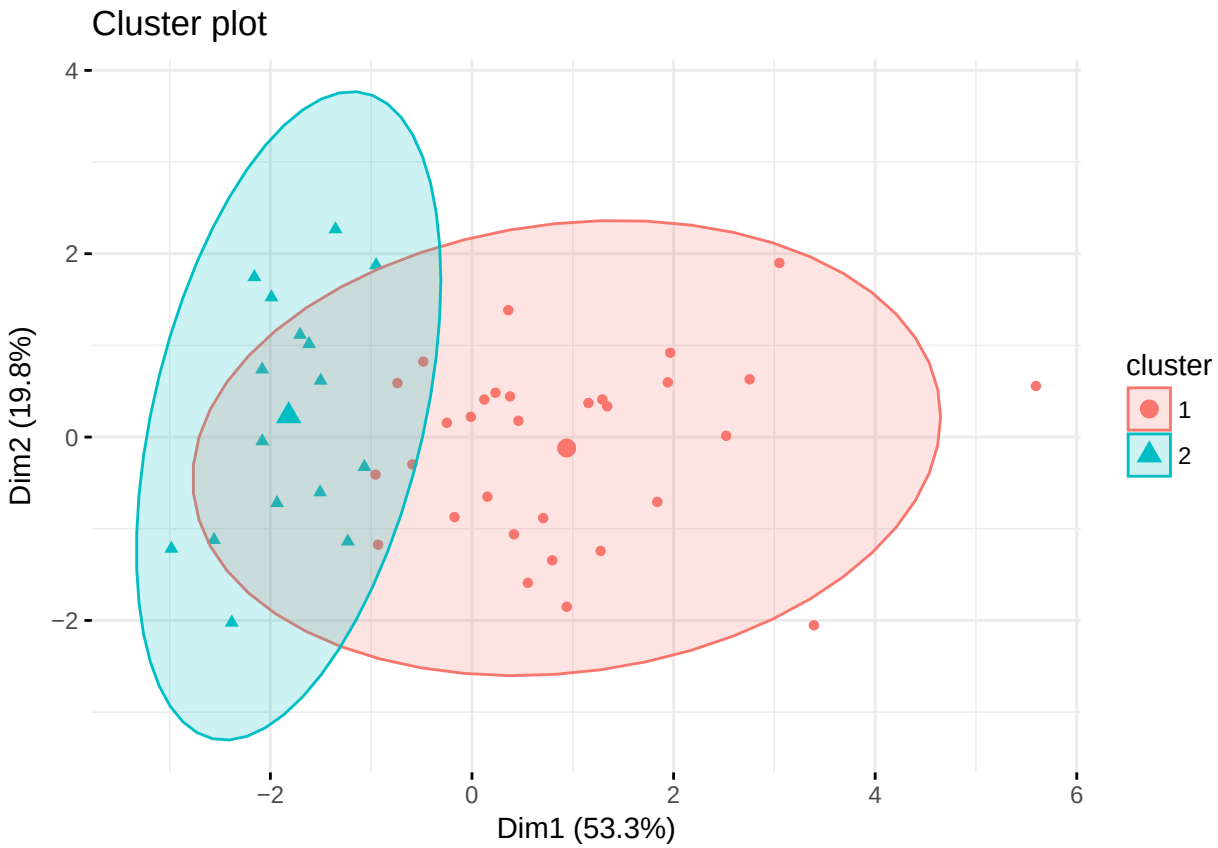

```
fviz_silhouette(km.res, palette = "jco", ggtheme = theme_classic())
```

```
##   cluster size ave.sil.width
## 1       1   12          0.34
## 2       2   16          0.57
## 3       3    5          0.52
## 4       4    3          0.34
## 5       5   11          0.40
```



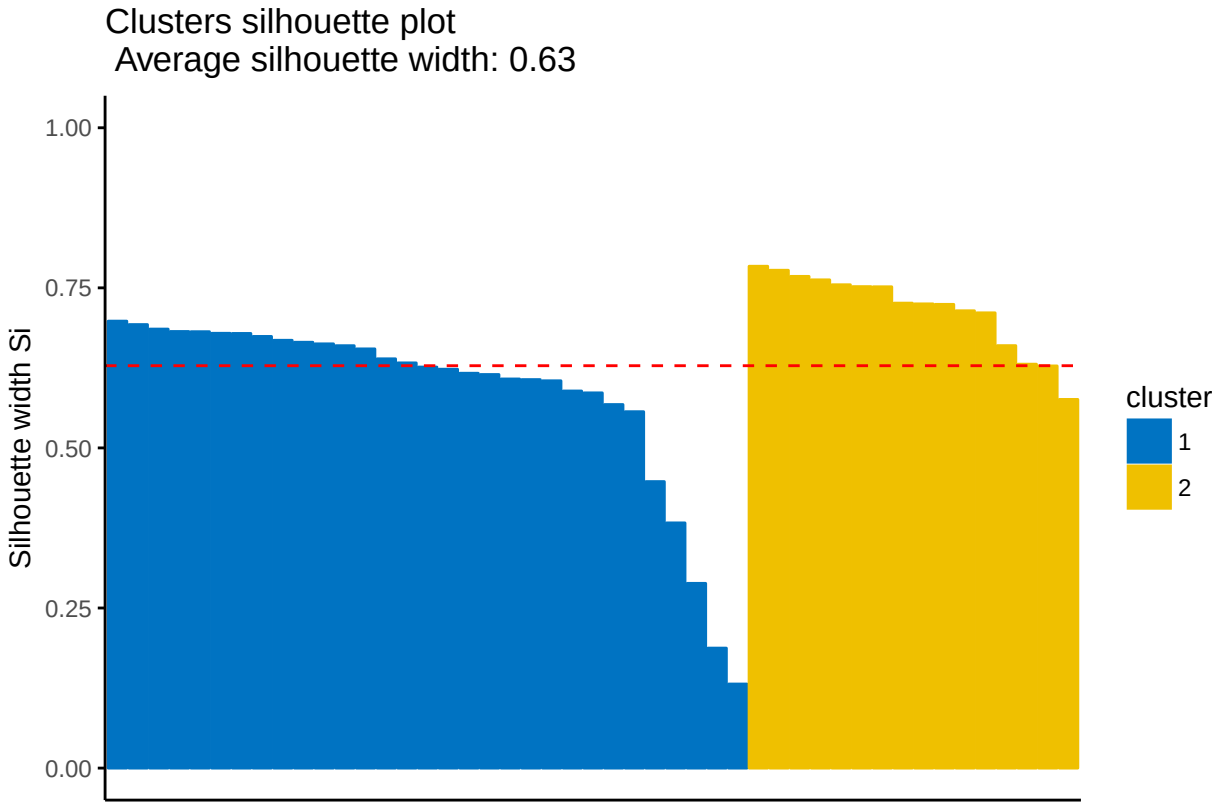
As the NbClust() function suggested, the real number of clusters may be 2. So instead of just relying on the GAP statistic, lets rely on all 30 methods for predicting the cluster number. Therefore, we will override the eclust function

```
km.res <- eclust(df, "kmeans", k = 2, nstart = 25, graph = FALSE)
fviz_cluster(km.res, geom = "point", ellipse.type = "norm", ggtheme = theme_minimal())
```




```
fviz_silhouette(km.res, palette = "jco", ggtheme = theme_classic())
```

```
##   cluster size ave.sil.width
## 1       1   31          0.58
## 2       2   16          0.72
```



With two clusters, the silhouette width drastically increases. Remember Observations with a large silhouette (1) are very well clustered. So we can now conclude that the `swiss` data set has 2 clusters!

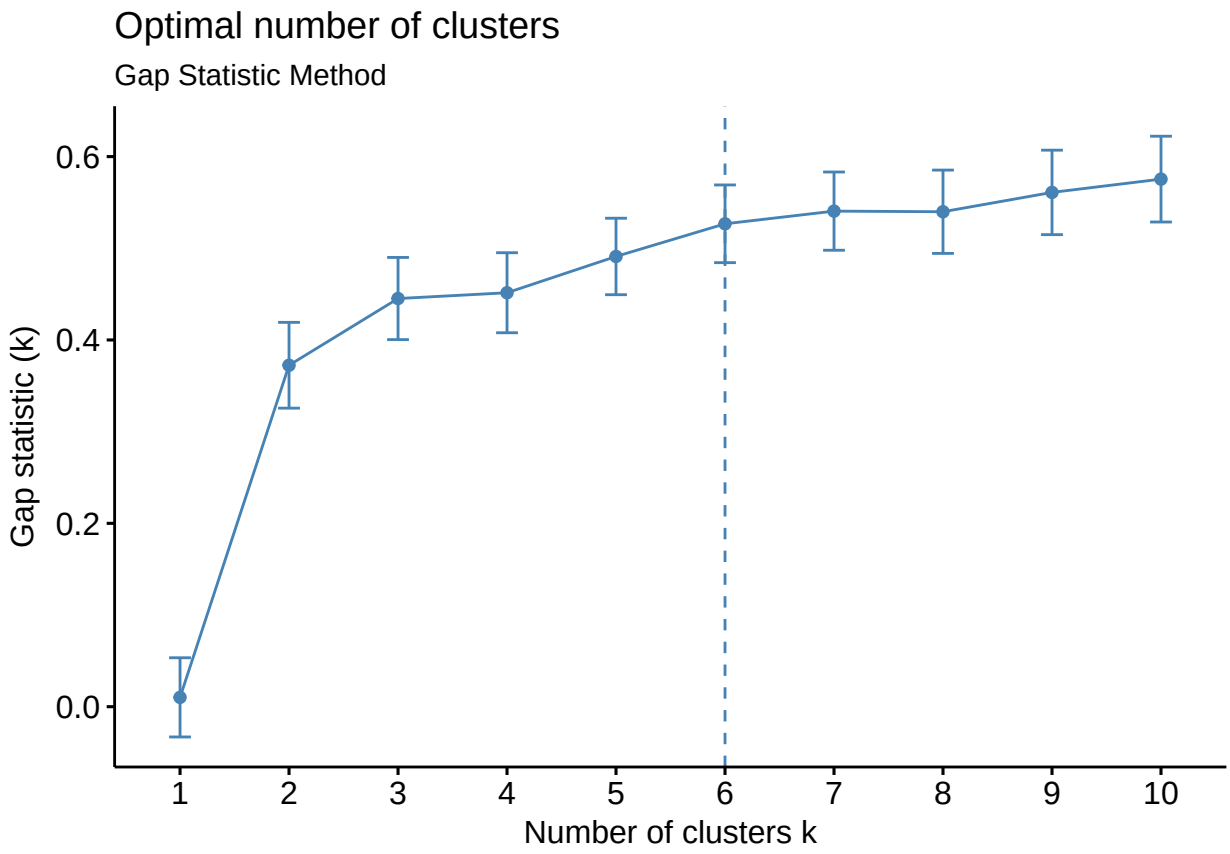
Part II: K-Mediods Approach

Recall that in k-means clustering, the ceter of a given cluster is calculated as the mean value of all the data points in the cluster. Using this method introduces the potential for large outliers to drastically influence the cluster mean.

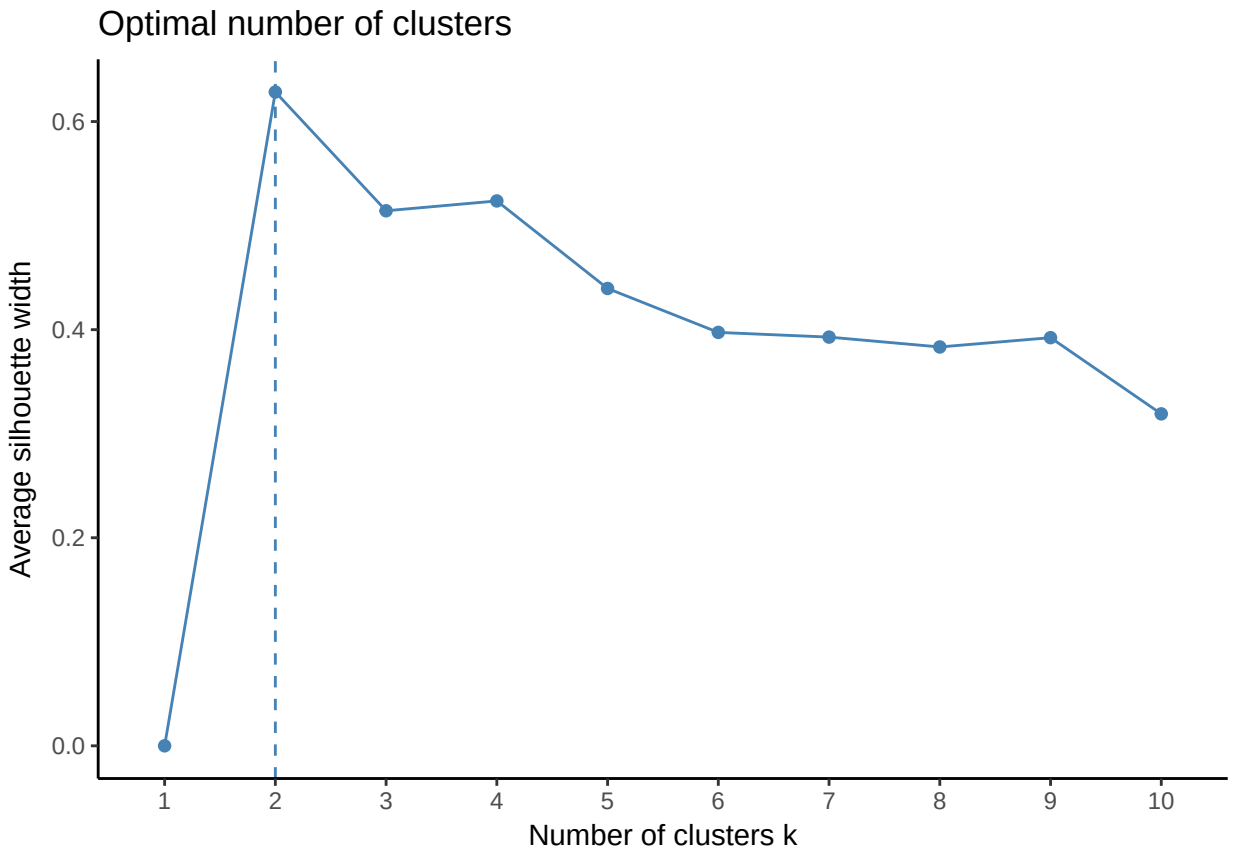
We can solve this using the k-mediod algorithm. A mediod refers to an object within a cluster for which the average dissimilarity between it and all the other points of the cluster is minimal. In other words, it is the most centrally located point in the cluster. Therefore it does not rely on taking the average. This means that, the algorithm is less sensitive to noise and outliers.

We have also been using euclidean distances in our clusters. If your data contains outliers Manhattan distances will give a more robust result. While Base R can calculate the PAM approach, the cluster package is much simpliler.

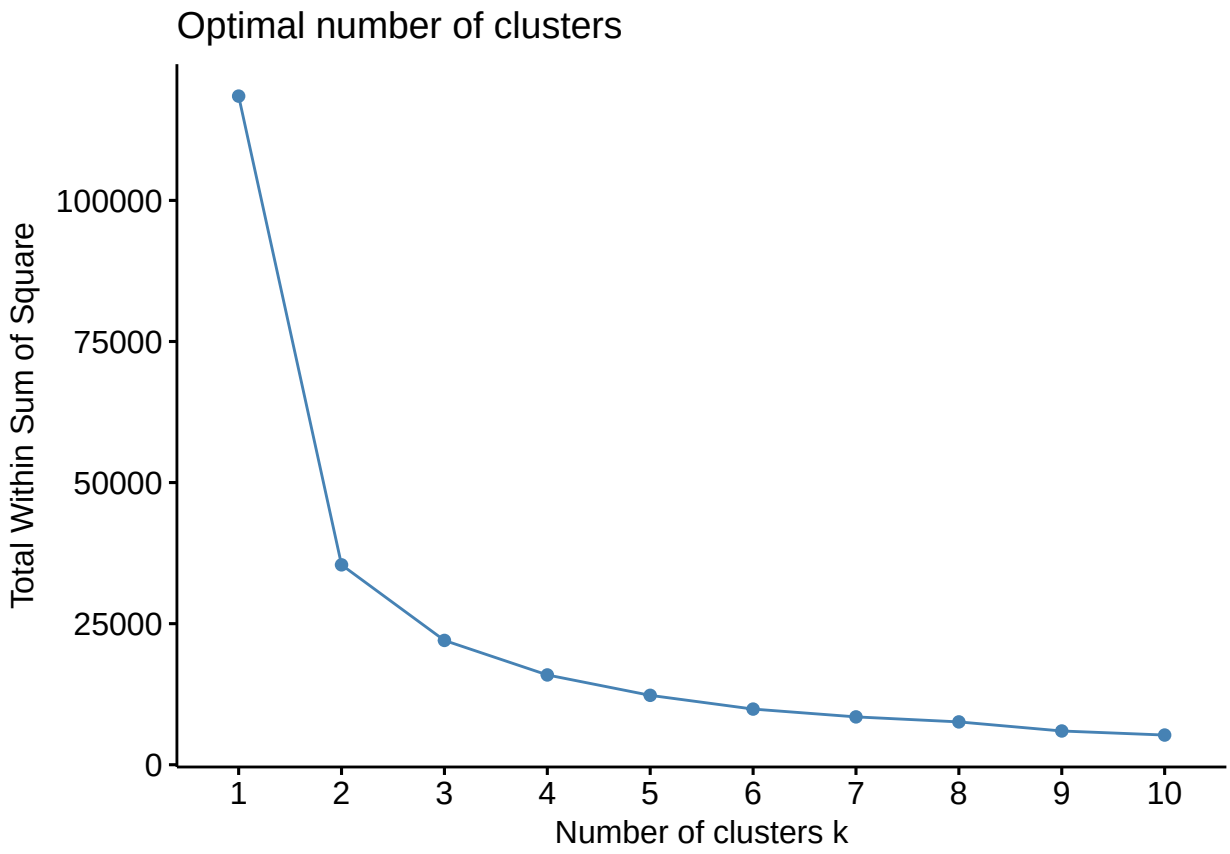
```
fviz_nbclust(x = df, FUNcluster = cluster::pam, method = "gap_stat", nboot = 500) +
  labs(subtitle = "Gap Statistic Method")
```




```
fviz_nbclust(df, cluster::pam, method = "silhouette") + theme_classic()
```




```
fviz_nbclust(df, FUNcluster = cluster::pam, method = "wss")
```




```
# For these functions we do not need to included the scaled data.  The algorithm will compute this usin
```

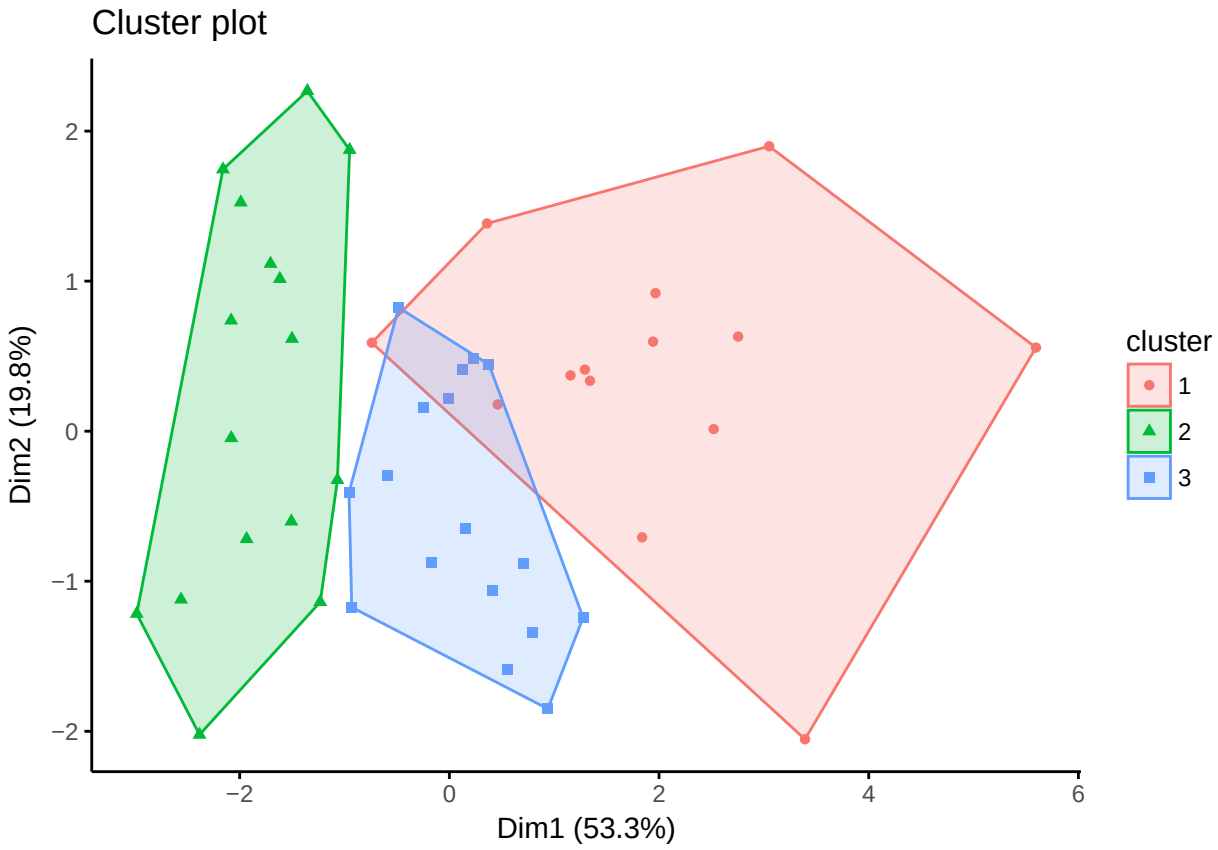
There doesn't seem to be great consensus as to what the optimal number of clusters is for the mediod approach. You could you similar methods above if you wanted to be more accurate. I will assume that the optimal number is 3. We can use the `pam` function in `cluster` to calculate the new clusters based on the mediod approach.

```
pam.res <- pam(df, 3)
print(pam.res)
```

```
## Medoids:
##         ID Fertility Agriculture Examination Education Catholic
## Vevey   29      58.3        26.8          25        19    18.46
## Monthey 35      79.4        64.9           7         3    98.22
## Morges  21      65.5        59.8          22        10     5.23
##         Infant.Mortality
## Vevey               20.9
## Monthey             20.2
## Morges              18.0
## Clustering vector:
##   Courtelary     Delemont Franches-Mnt      Moutier   Neuveville
##            1            2            2            1            3
##   Porrentruy        Broye        Glane      Gruyere       Sarine
##            2            2            2            2            2
##      Veveyse        Aigle      Aubonne     Avenches     Cossonay
##            2            3            3            3            3
##    Echallens     Grandson     Lausanne    La Vallee       Lavaux
##            3            1            1            1            3
```

```
##       Morges       Moudon        Nyone         Orbe         Oron
##            3            3            3            3            3
##      Payerne Paysd'enhaut       Rolle       Vevey      Yverdon
##            3            3            3            1            3
##      Conthey    Entremont      Herens     Martigwy      Monthey
##            2            2            2            2            2
##   St Maurice       Sierre        Sion       Boudry La Chauxdfnd
##            2            2            2            1            1
##     Le Locle    Neuchatel  Val de Ruz ValdeTravers V. De Geneve
##            1            1            3            1            1
##  Rive Droite  Rive Gauche
##            1            1
## Objective function:
##    build     swap
## 21.94562 19.58769
##
## Available components:
##  [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"
##  [6] "clusinfo"   "silinfo"    "diss"       "call"       "data"
```

```
fviz_cluster(pam.res, df, stand = TRUE, geom = "point", repel = TRUE, ggtheme = theme_classic())
```



Approach #2: Hierarchical Clustering

In contrast to partitioning clustering, hierarchical clustering does not required to pre-specify the number of clusters.

Agglomerative clusting (AGNES): Each observation is initially considered as a cluster of its own leaf. Then the most similar clusters are successively merged. In this way, agglomerative clustering works in a "bottom up" fashion.

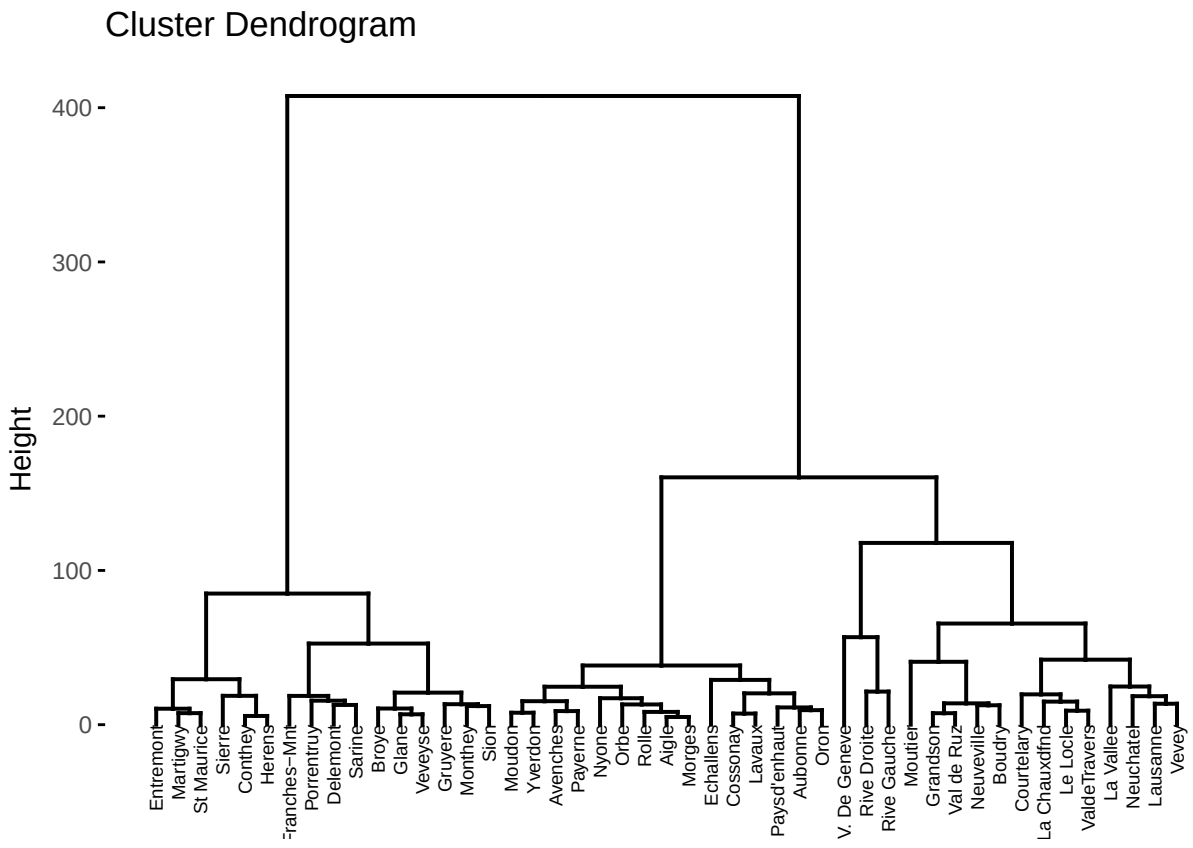
Steps to Agglomerative Hierarchical Clustering

1.) Compute disimilarity between every pair of objects 2.) Determine Linkage - Take the distance information and group pairs into clusters until all obkects are linked together in a hierarchical tree 3.) Determine where to cut the hierarchical tree into clusters.

- The data should be numeric!

Agglomerative Clustering In Base R

```
res.dist <- dist(swiss, method = "euclidean") #Step One - Disimilarity
res.hc <- hclust(d = res.dist, method = "ward.D2") #Step Two - Linkage (Ward minimizes Within Cluster V
fviz_dend(res.hc, cex = 0.5)
```



Validation of Clusters

After linking the objects, it is wise to verify the tree. Cophenetic Distance is a measure of how faithfully a dendrogram preserves the pairwise distances between the original unmodeled data points. Values above 0.75 are considered good.

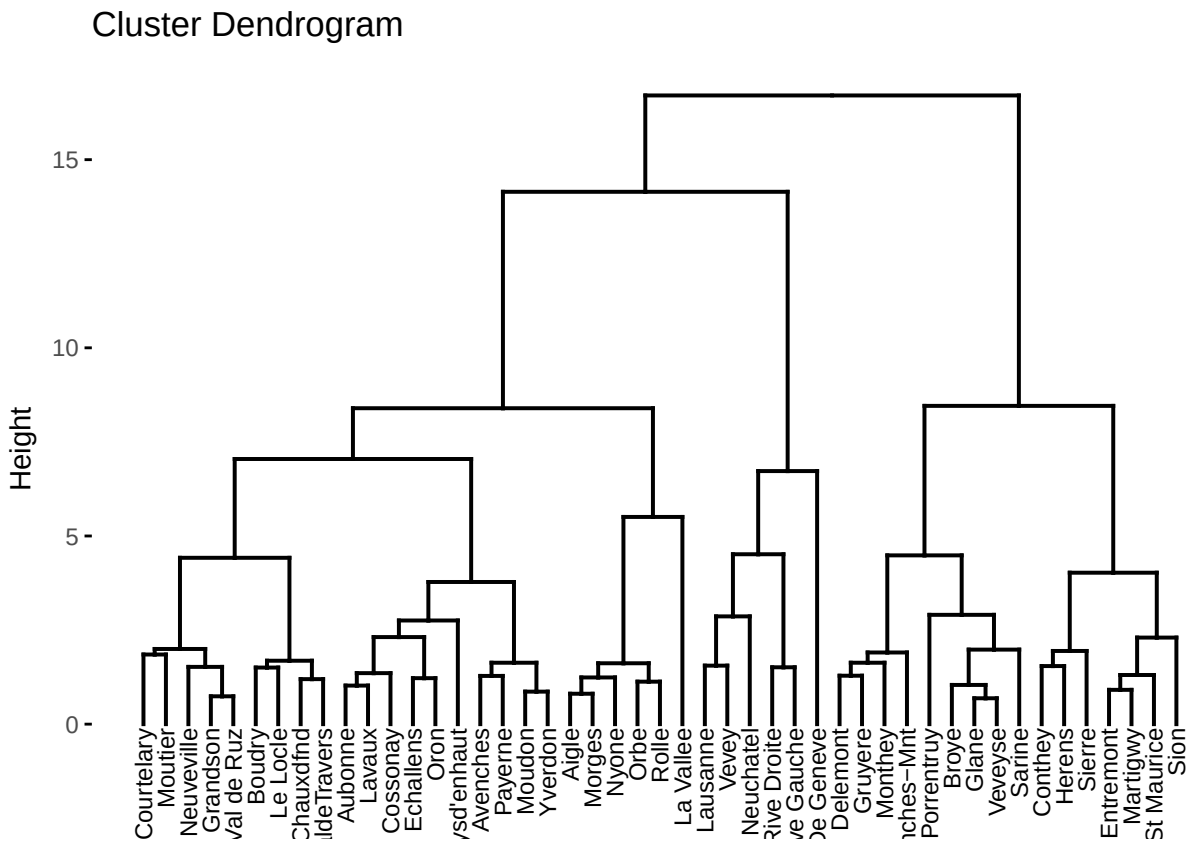
```
res.coph <- cophenetic(res.hc)
cor(res.dist, res.coph)
```

```
## [1] 0.9140627
```

Agglomerative Clustering Using `Cluster`

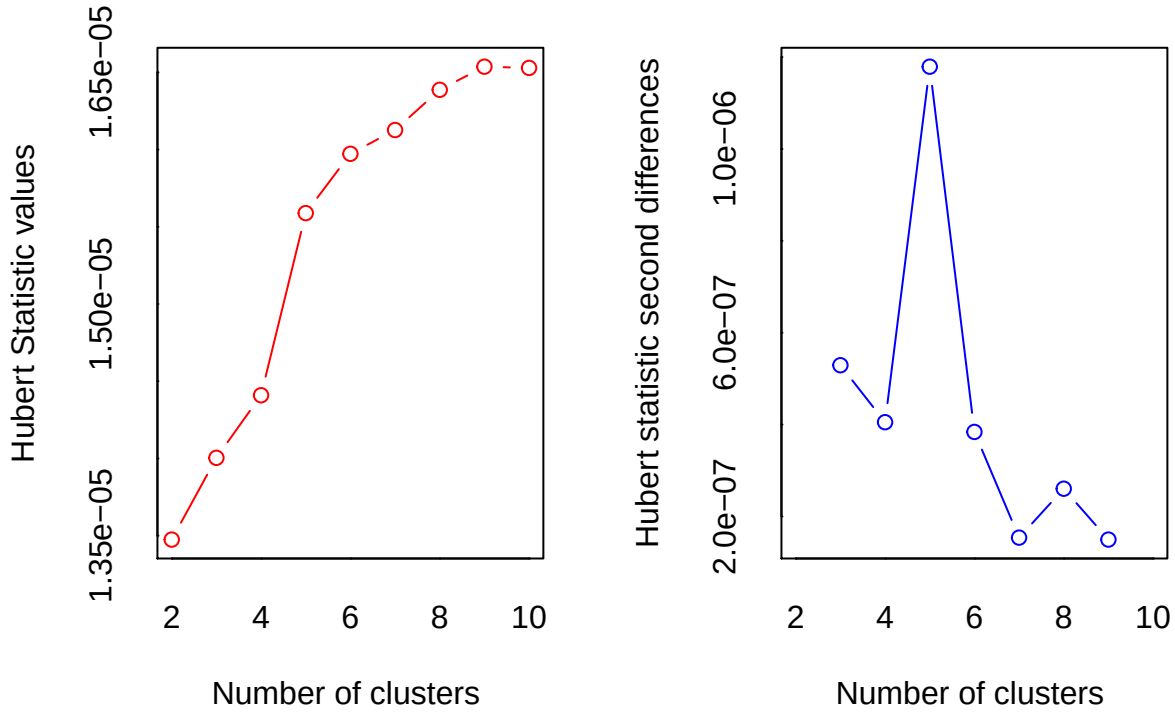
As you saw, there were three steps to agglomerative clustering using base R. THe Cluster Packages condenses these steps to just one.

```
res.agnes <- agnes(x = df, #data frame
                   stand = TRUE, #Standardize the Data
                   metric = "euclidean", # Metric for Distance
                   method = "ward") #Linkage Method

fviz_dend(res.agnes, cex = 0.6, rect = TRUE) # Now we don't need long code to cut the tree in order to
```

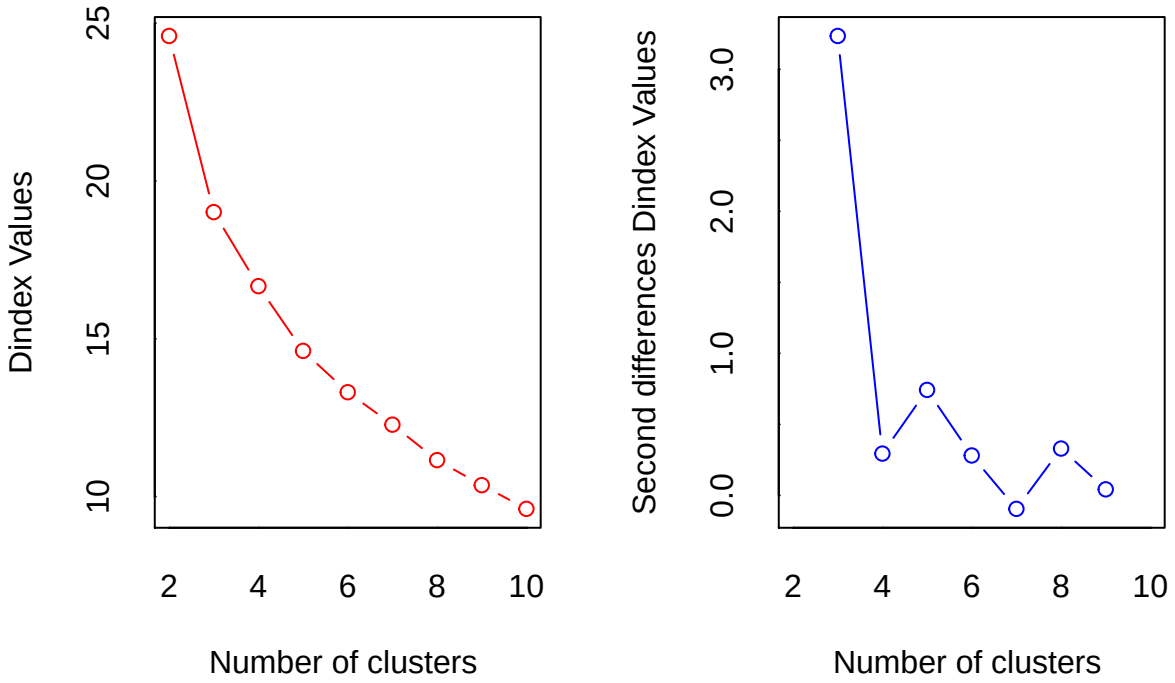


As you may have noticed, the agnes function does not tell us the number of clusters. However we can rely on the NbClust() package.

```
nb <- NbClust(df, distance = "euclidean", min.nc = 2,
              max.nc = 10, method = "ward.D2")
```




```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```



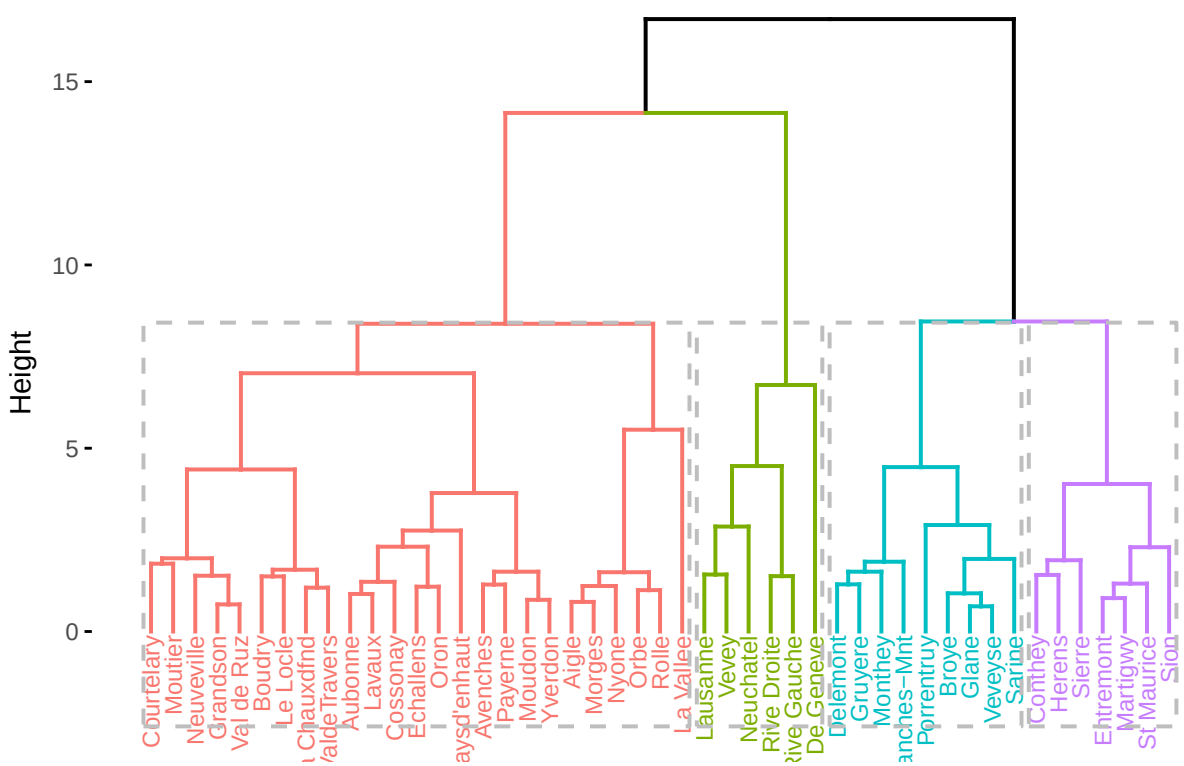

```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
```

```
##
## *******************************************************************
## * Among all indices:
## * 10 proposed 2 as the best number of clusters
## * 5 proposed 3 as the best number of clusters
## * 3 proposed 4 as the best number of clusters
## * 2 proposed 6 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 3 proposed 10 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

There are 4 different clusters, now we can work do make the dendogram visually appealing, with the clusters clearly defined.

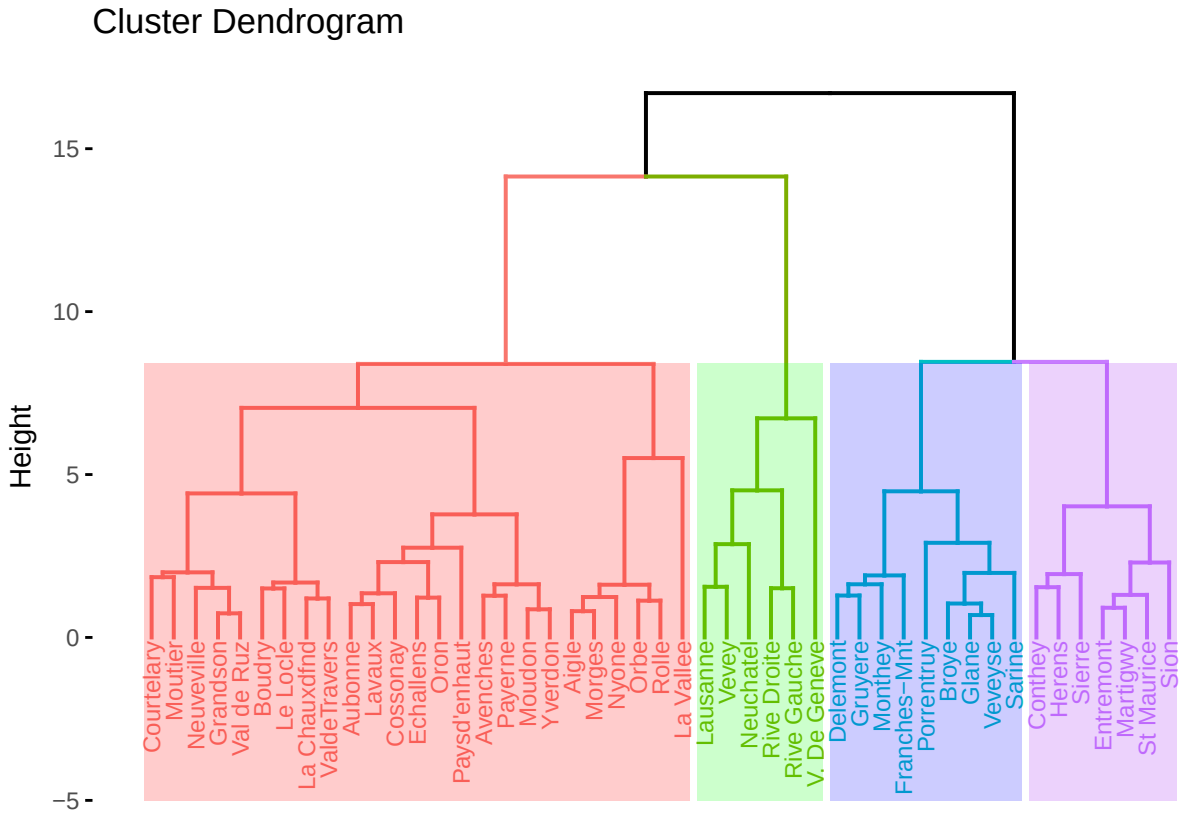
Visualizing Agglomerative CLustering

```
fviz_dend(res.agnes, cex = 0.6, k = 4, rect = TRUE)
```

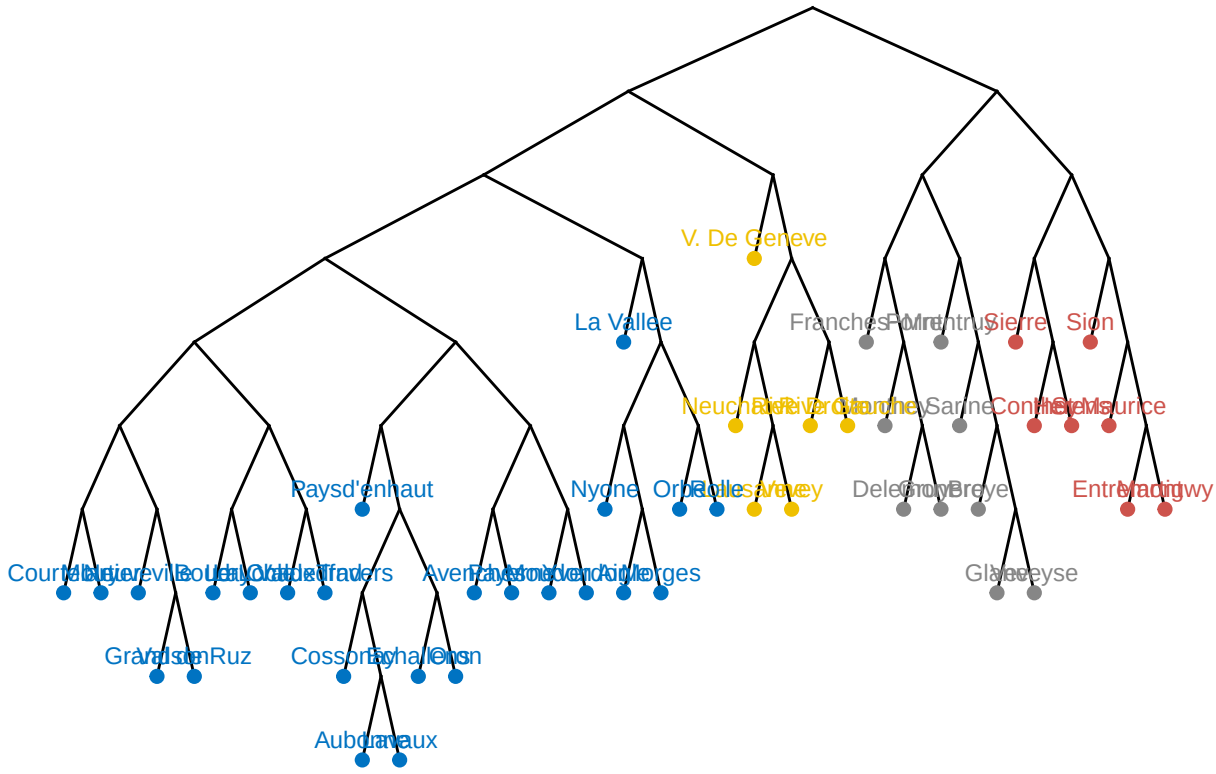



```
fviz_dend(res.agnes, cex = 0.6, k =4, rect = TRUE, color_labels_by_k = TRUE, rect_border = c("red", "gr
```

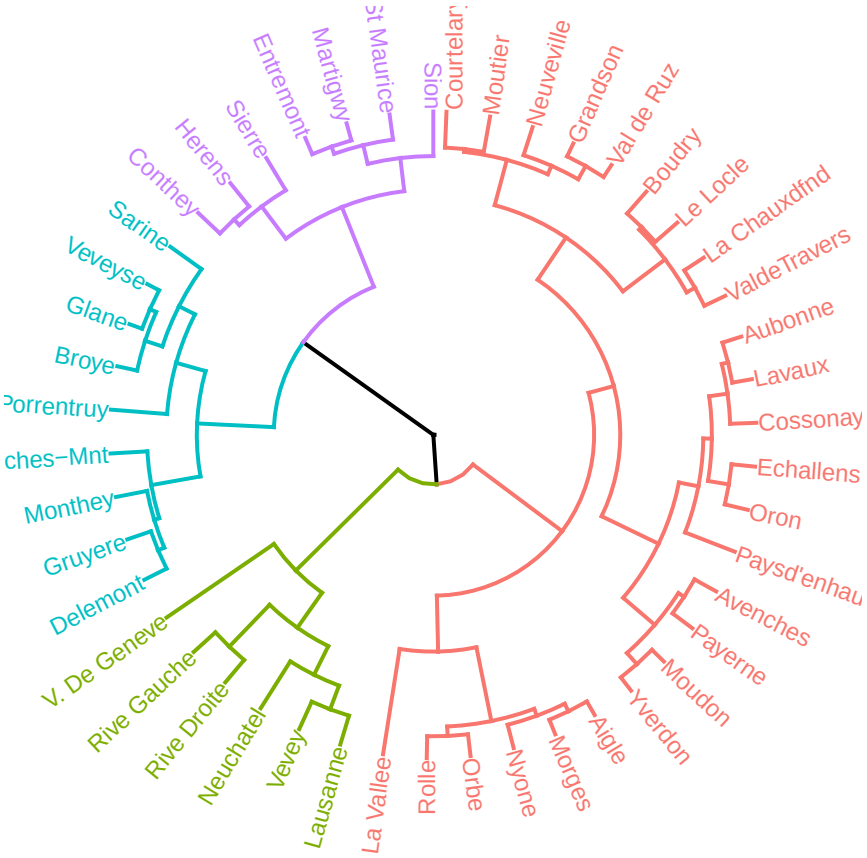
```
## Warning in if (color == "cluster") color <- "default": the condition has
## length > 1 and only the first element will be used
```




```
fviz_dend(res.agnes, k = 4, k_colors = "jco", type = "phylogenic", relep = TRUE , phylo_layout = "layout
```



```
fviz_dend(res.agnes, cex = 0.6, k = 4, type = "circular", rect = TRUE)
```



Exporting to PDF

```
pdf("dendrogram.pdf", width = 10, height = 15)
p <- fviz_dend(res.agnes, cex = 0.6, k = 4, type = "circular", rect = TRUE)
```

Calculating P-Values

```
library(pvclust)
pval <- parPvclust(cl = NULL, df, method.hclust = "ward", method.dist = "euclidean", nboot = 200, iseed
```

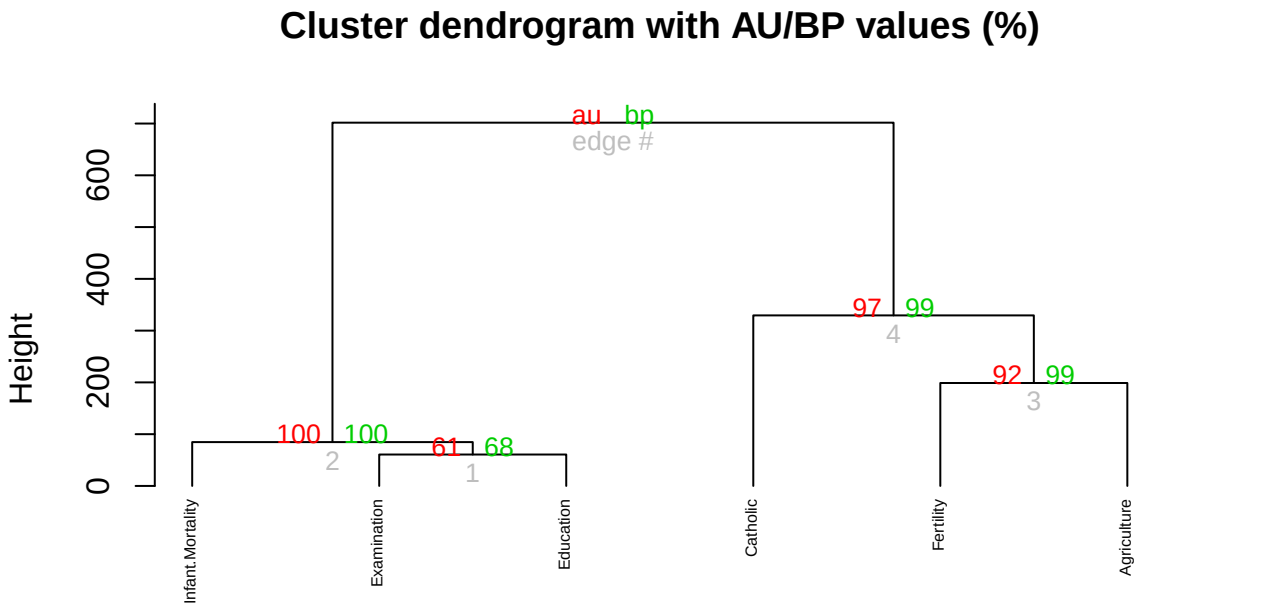
```
## Warning in parPvclust(cl = NULL, df, method.hclust = "ward", method.dist = "euclidean", : "parPvclus
## It is available for back compatibility but will be unavailable in the future.
```

```
## Warning in pvclust.parallel(cl = cl, data = data, method.hclust =
## method.hclust, : Cluster size is too small (or NULL). non-parallel version
## is executed
```

```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

```
## Bootstrap (r = 0.49)... Done.
## Bootstrap (r = 0.6)... Done.
## Bootstrap (r = 0.68)... Done.
## Bootstrap (r = 0.79)... Done.
## Bootstrap (r = 0.89)... Done.
## Bootstrap (r = 1.0)... Done.
## Bootstrap (r = 1.09)... Done.
## Bootstrap (r = 1.19)... Done.
## Bootstrap (r = 1.3)... Done.
## Bootstrap (r = 1.38)... Done.
```

```
plot(pval, hang = -1, cex = 0.5)
```



Values on the dendrogram are the Approximately Unbiased P-Values(Red,left). These are similar to bootstrap values but are more technical. The Bootstrap P-values are more commonly known (green, right). In this case, pvclust will bootstrap the columns of the dataset. An important note is that if you would like to bootstrop the rows, you can transpose the data.

For example a BP value of 55 indicates that these 8 genes ended up in the same cluster 55 runs out of the 100 bootstraps. If the BP is high then the cluster can be supported by the data.