Clustering

R. Porcedda, F. Chiaromonte (special thanks to J. Di Iorio and L. Insolia)

February 5th 2026

Contents

Introduction

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (according to a similarity or dissimilarity measure) to each other than to those in other groups (clusters). Thus, clustering looks to find homogeneous subgroups among the objects.

Note that we can cluster observations on the basis of the features, or features on the basis of the observations. We will focus on clustering observations on the basis of the features.

Cluster analysis itself is not one specific algorithm, but the general unsupervised classification ("no label") task to be solved. It can be achieved by various algorithms. We are going to focus on exhaustive algorithms which determine a hard partition: every point belongs to one group, and one group only.

Libraries

We are going to use

- **cluster**: *Finding Groups in Data*
- **NbClust**: *Determining the Relevant Number of Clusters in a Data Set*

- **factoextra**: *Extract and Visualize the Results of Multivariate Data Analyses*

```
# do not forget to install them first! e.g.,
# install.packages("cluster")
# or something automated such as:
# if(!require(cluster)){install.packages("cluster"); library(cluster)}

library(cluster)       # methods for Cluster analysis
library(factoextra)    # to extract and visualize the output of  multivariate analyses
library(NbClust)       # to determine the optimal number of clusters
```

Data

Today we are going to use the *iconic* **Anderson's Iris data set** (you can find more details on Wikipedia).

The data set consists of 50 samples from three species of iris flowers (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample:

- sepal length & width
- petal length & width

which are all measured in centimeters.

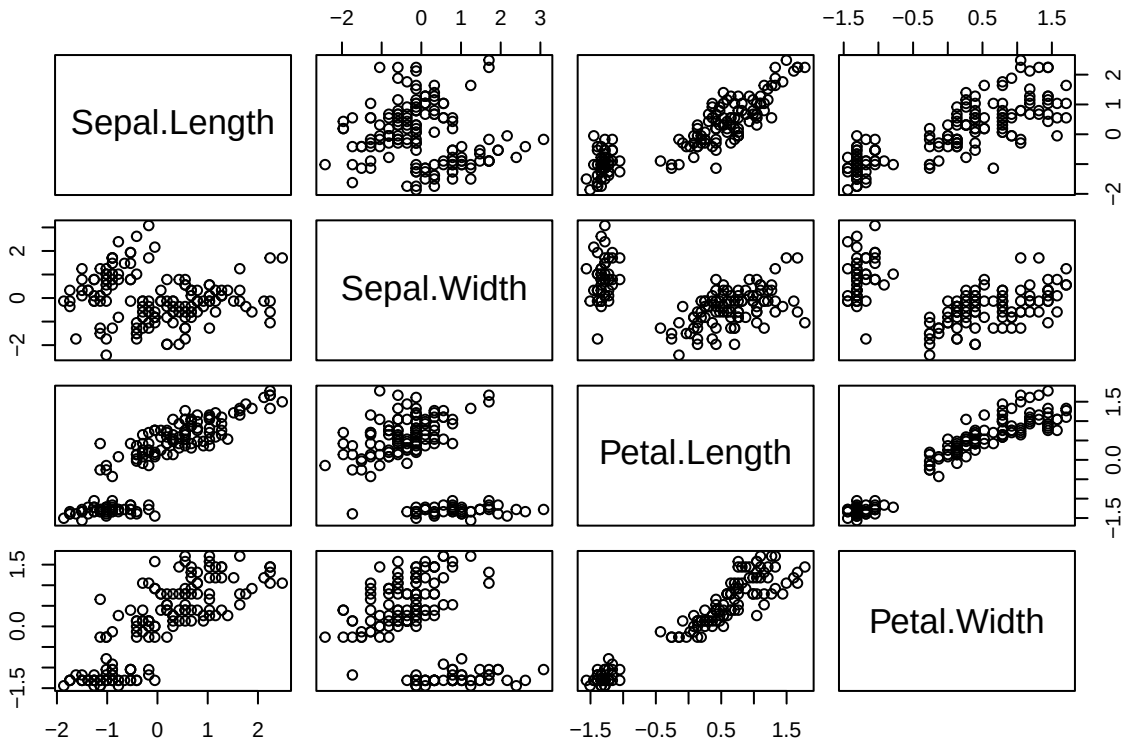
The Iris dataset is available in several R packages. You might want to give a look at the **datasets** package.

```
library(utils)                      # a general utility package
# ?data()                           # list all available data sets across all loaded packages
# try(data(package = "cluster"))    # list the data sets in the cluster package

library(datasets)
help(datasets)
# library(help = "datasets")        # full list of datasets
help(iris)
data("iris")
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

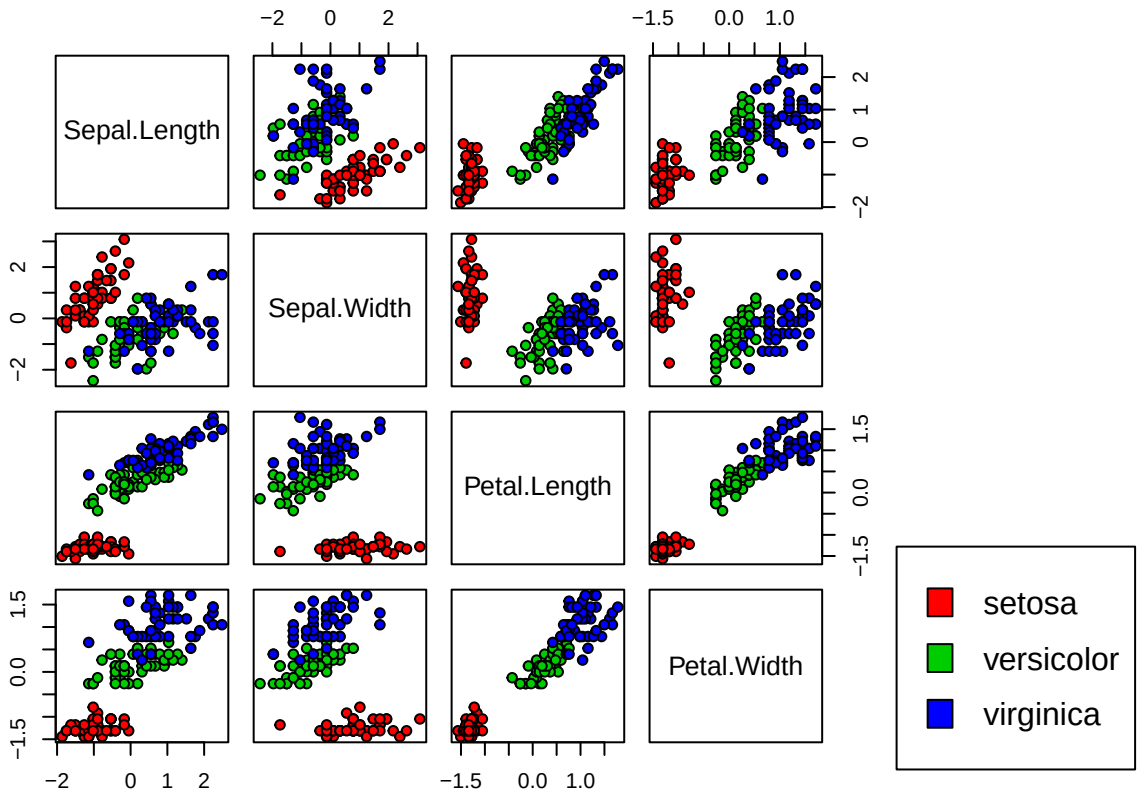
Let us remove the *Species* label and plot the data in a pairwise scatterplot (**pairs** command). How many clusters are present?

```
iris4 <- iris[,1:4]   # Excluding the "Species" label in column 5
iris4 <- scale(iris4) # Standardize (column-wise)
pairs(iris4)
```



This is the scatterplot with species labels.

```
# all the extra commands are used just to include a legend out of the figure panel
pairs(iris4, main="Iris Data (red=setosa,green=versicolor,blue=virginica)",
      pch=21, bg=c("red","green3","blue")[unclass(iris$Species)],
      oma=c(3,3,3,15))
par(xpd = TRUE)
legend("bottomright", fill = c("red","green3","blue"), legend = c( levels(iris$Species)))
```



Hierarchical Clustering

These algorithms do not provide a unique data partition, but rather a nested hierarchical structure – clusters are merged according to certain distances. The hierarchy is usually represented by a dendrogram. The y-axis marks the "similarity" across groups – i.e., the distance at which clusters are merged – and the x-axis contains a permutation of the grouped objects in such a way that clusters do not mix (i.e., vertical lines to not cross each other).

Pros:

- It does not require us to pre-specify the number of clusters $\boldsymbol{k}$
- Dendrogram results in an attractive and simple tree-based representation of the observations

Cons:

- Often the choice of where to cut the dendrogram to select clusters is not so clear
- The assumption of hierarchical structure might be unrealistic (i.e. clusters obtained by cutting the dendrogram at a given height are necessarily nested within the clusters obtained by cutting the dendrogram at any greater height)

Strategies for hierarchical clustering generally fall into two types:

- Agglomerative Hierarchical Clustering
- Divisive Hierarchical Clustering

Agglomerative Hierarchical Clustering

It is a "bottom-up" approach which generates a sequence of nested partitions of the data – progressively less granular:

1. Start with n clusters: each containing one data point ($k = n$)
2. At each iteration:
   - Find the "closest pair" of clusters
   - Merge them and update the list of clusters
   - Update the matrix of cluster distances (it captures their similarity/dissimilarity)
3. Iterate until all data points belong to a single cluster (i.e., a giant group with $k = 1$)

To perform the Agglomerative Hierarchical Clustering we can use the basic function **hclust**.

```
help(hclust)
```

We can see that the function requires:

- ***d***: a dissimilarity structure
- **method**: the linkage method to be used.

A dissimilarity matrix based on the **Euclidean distance** can be produced by **dist** in the following way. *Note*: in principle this is a symmetric matrix of size $n \times n$ with zero entries on the main diagonal.

```
eu_iris <- dist(iris4, method='euclidean')
class(eu_iris) # note: this is not a matrix, but an object of a specific class
```

```
## [1] "dist"
```

```
dim(as.matrix(eu_iris))    # we can transform it into the n-by-n matrix of distances
```

```
## [1] 150 150
```

```
as.matrix(eu_iris)[1:5,1:5] # distances between the first 5 observations
```

```
##           1         2         3         4         5
## 1 0.0000000 1.1722914 0.8427840 1.0999999 0.2592702
## 2 1.1722914 0.0000000 0.5216255 0.4325508 1.3818560
## 3 0.8427840 0.5216255 0.0000000 0.2829432 0.9882608
## 4 1.0999999 0.4325508 0.2829432 0.0000000 1.2459861
## 5 0.2592702 1.3818560 0.9882608 1.2459861 0.0000000
```

Now we are ready to create the hierarchical structure, based on the following linkage methods.

```
hc_single   <- hclust(eu_iris, method='single')     # single linkage
hc_complete <- hclust(eu_iris, method='complete')   # complete linkage
hc_average  <- hclust(eu_iris, method='average')    # average linkage
hc_centroid <- hclust(eu_iris, method='centroid')   # centroid linkage

str(hc_single)            # it's a list
```

```
## List of 7
##  $ merge      : int [1:149, 1:2] -102 -8 -11 -10 -1 -129 -41 -128 -28 -3 ...
##  $ height     : num [1:149] 0 0.121 0.121 0.131 0.131 ...
##  $ order      : int [1:150] 107 61 99 58 94 109 63 119 88 69 ...
##  $ labels     : NULL
##  $ method     : chr "single"
##  $ call       : language hclust(d = eu_iris, method = "single")
```

```
##  $ dist.method: chr "euclidean"
##  - attr(*, "class")= chr "hclust"
```

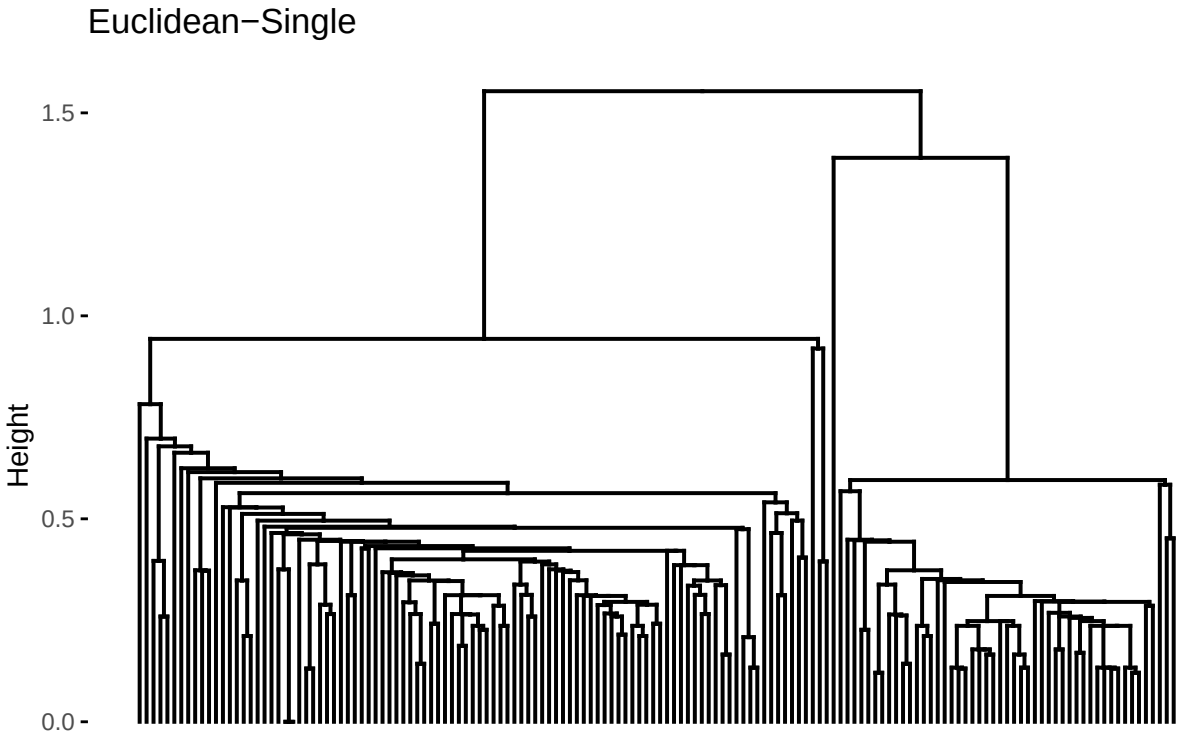
```
head(hc_single$merge, 10) # see the first 10 aggregations
```

```
##       [,1] [,2]
##  [1,] -102 -143
##  [2,]   -8  -40
##  [3,]  -11  -49
##  [4,]  -10  -35
##  [5,]   -1  -18
##  [6,] -129 -133
##  [7,]  -41    5
##  [8,] -128 -139
##  [9,]  -28    7
## [10,]   -3  -48
```

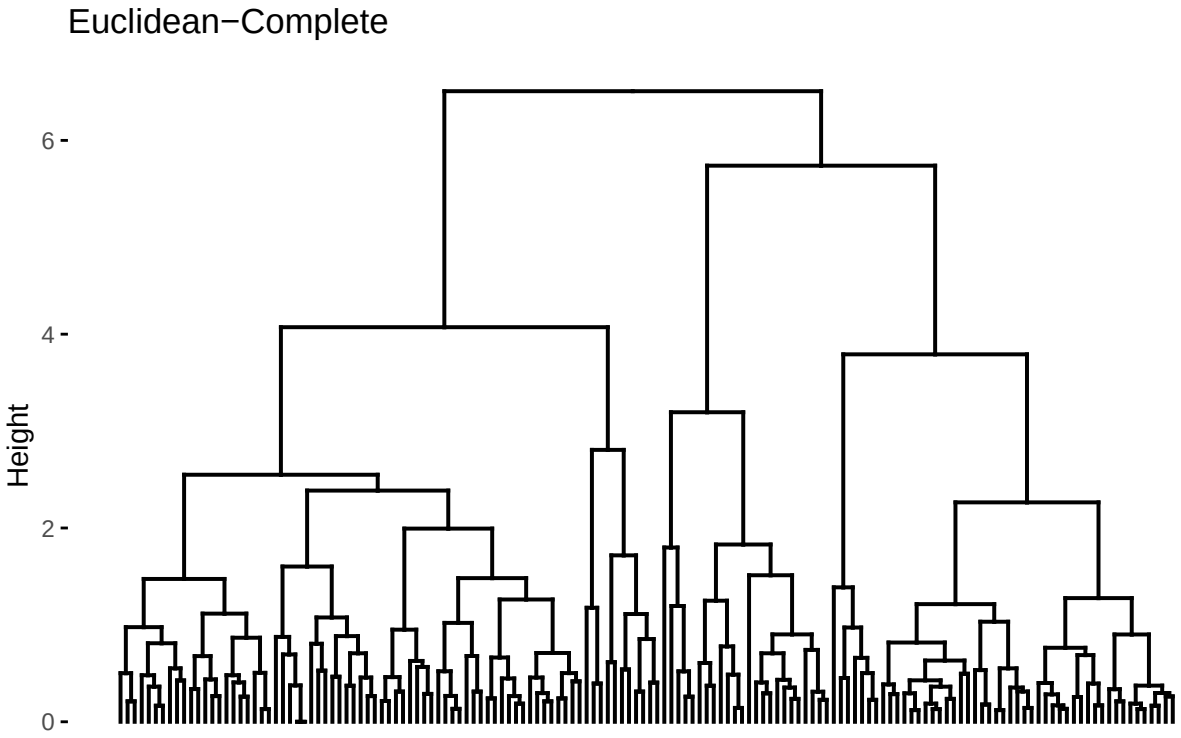
```
# to see its interpretation, check the "Value" (i.e., output) paragraph in here:
#help(hclust)

# to check that obs 102 and 143 are merged first
#as.matrix(eu_iris)[102,143]  # look that the distance between 102 and 143 is 0
#iris[c(102,143),]  # they report the same values for all covariates
```

The hierarchy is represented through a dendrogram: a plot illustrating a sequence of data partitions into clusters.

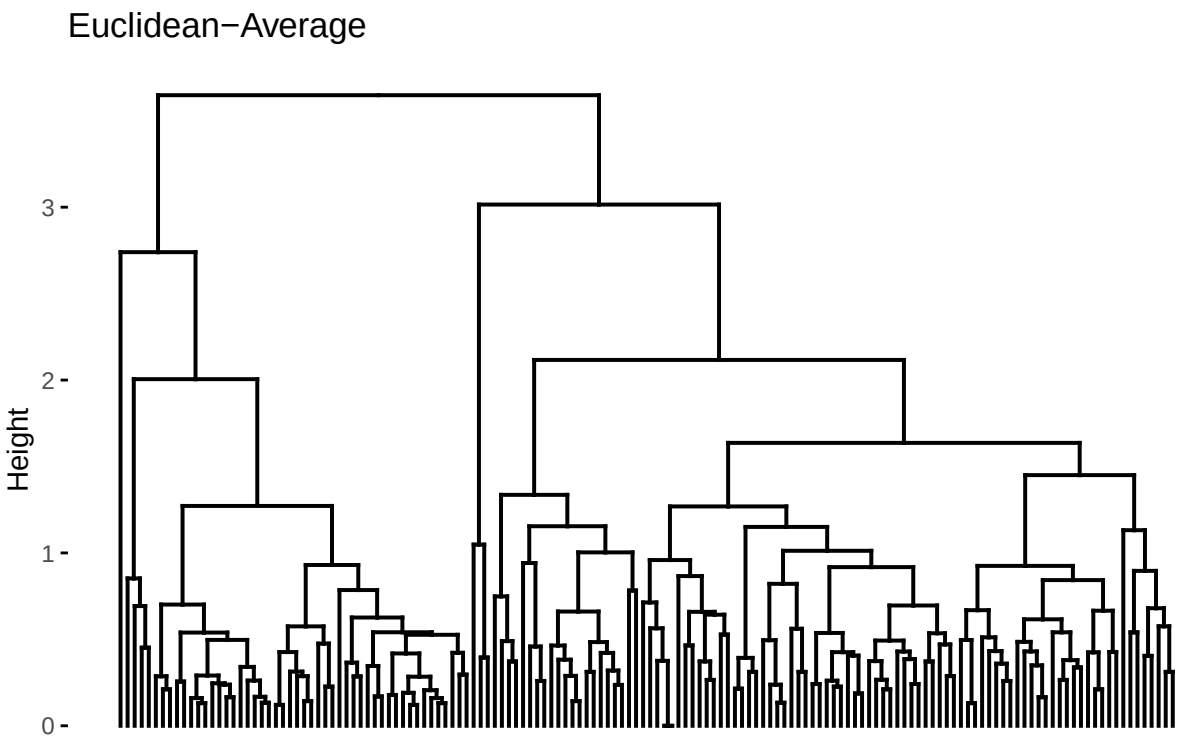
```
  # par(mfrow=c(2,2))
  fviz_dend(hc_single,   as.ggplot = TRUE, show_labels = FALSE, main='Euclidean-Single')
```



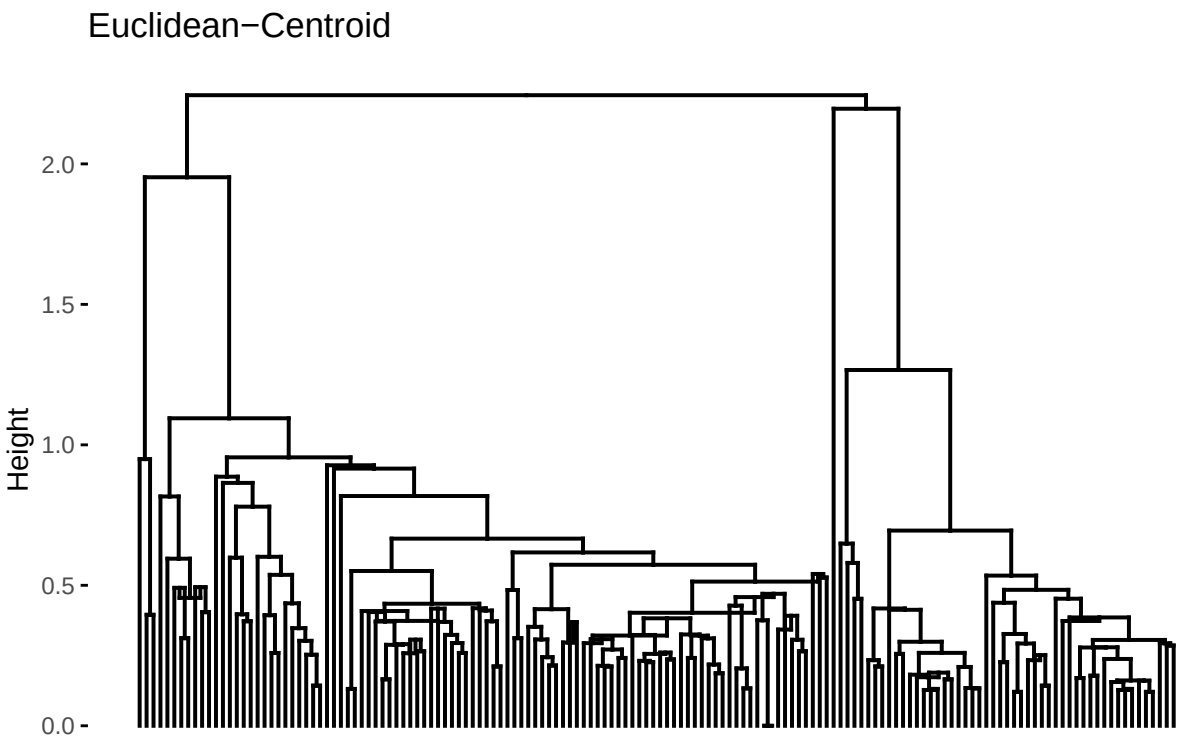

```
fviz_dend(hc_complete, as.ggplot = TRUE, show_labels = FALSE, main='Euclidean-Complete')
```




```
fviz_dend(hc_average,  as.ggplot = TRUE, show_labels = FALSE, main='Euclidean-Average')
```




```
fviz_dend(hc_centroid, as.ggplot = TRUE, show_labels = FALSE, main='Euclidean-Centroid')
```



To retrieve any given clustering solution, we have to cut the dendrogram using the **cutree** command.

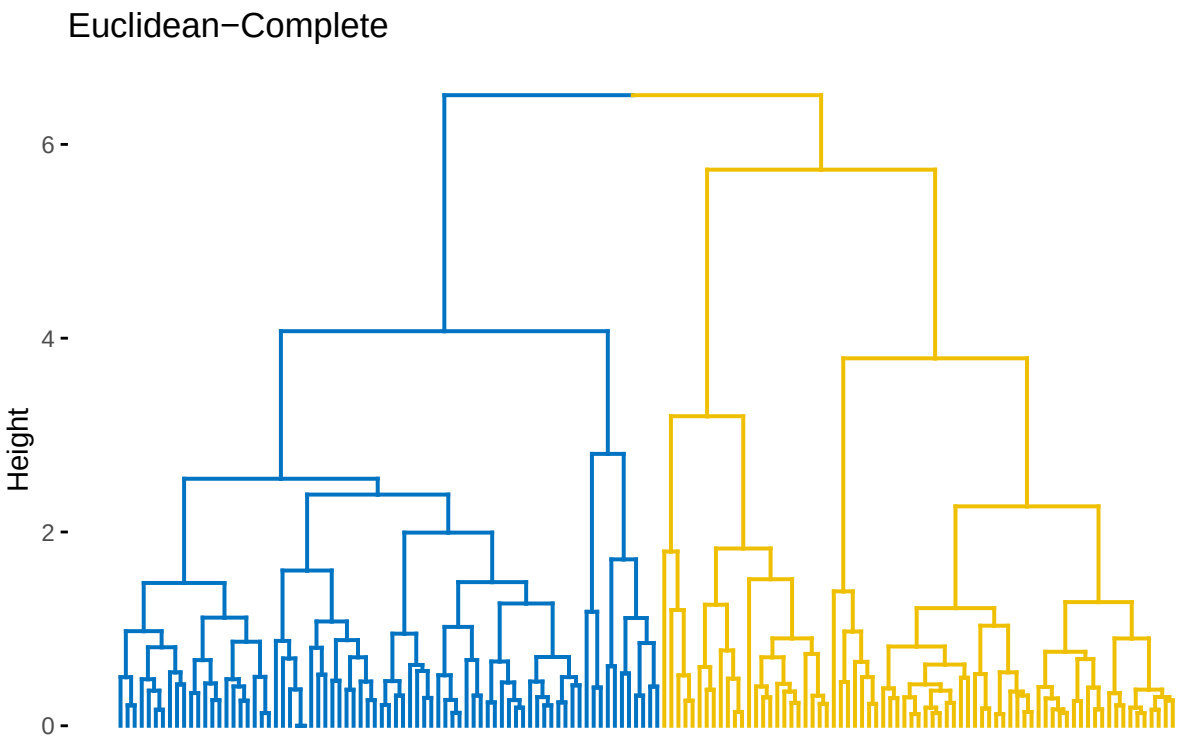
The cut can be performed according to:

- ***k***: an integer scalar or vector (if you want more than one partition) with the desired number of groups

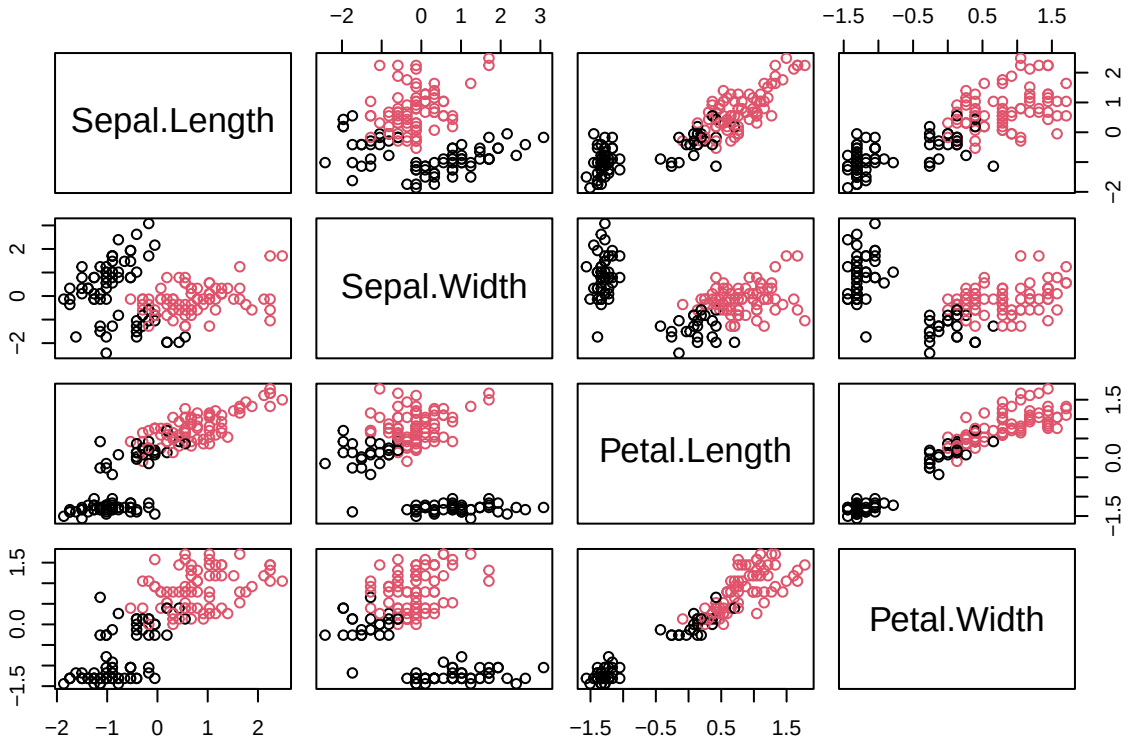
```
cluster_k <- cutree(hc_complete, k = 2)      # identify 2 groups
cluster_k
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 1 2 1 2 1 1 2 1 2 2 2 2 1 1 1 2 2 2 2
##  [75] 2 2 2 2 2 1 1 1 1 2 2 2 2 1 2 1 1 2 1 1 1 2 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2
```

```
fviz_dend(hc_complete, k = 2, k_colors = "jco",
          as.ggplot = TRUE, show_labels = FALSE,
          main='Euclidean-Complete')
```




```
pairs(iris4, col=cluster_k)                    # pairwise scatterplot with clusters
```

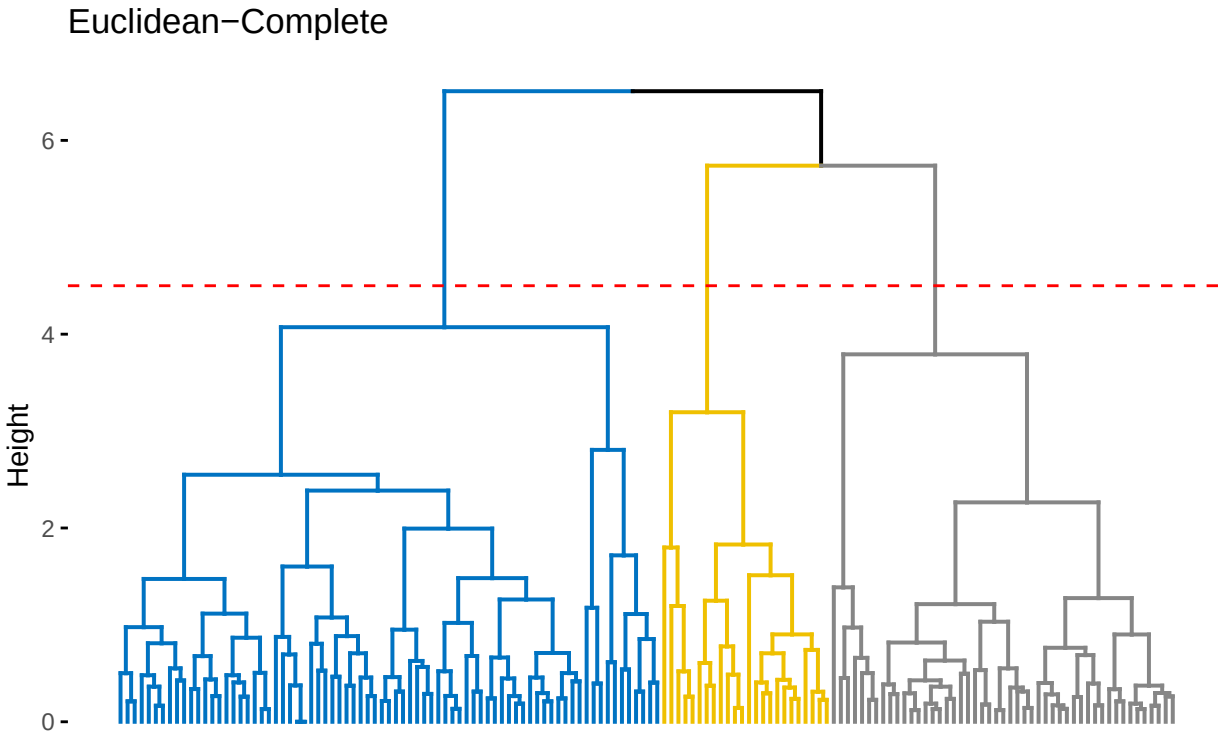


- ***h***: numeric scalar or vector with heights where the tree should be cut

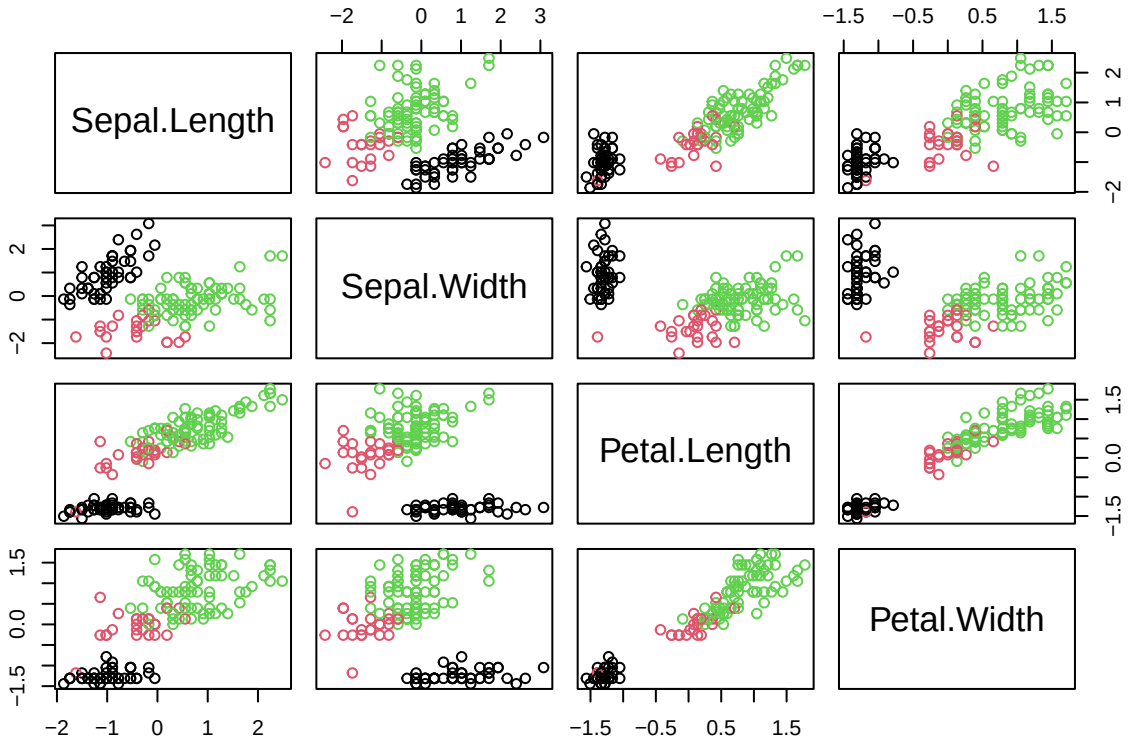
```
clHeight <- 4.5
cluster_h <- cutree(hc_complete, h = clHeight)    #identify groups below a certain height
cluster_h
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 2 1 1 1 1 1 1 1 1 3 3 3 2 3 2 3 2 3 2 2 3 2 3 3 3 3 2 2 2 3 3 3 3
##  [75] 3 3 3 3 3 2 2 2 2 3 3 3 3 2 3 2 2 3 2 2 2 3 3 3 2 2 3 3 3 3 3 3 2 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

```
fviz_dend(hc_complete, h = clHeight, k_colors = "jco",
          as.ggplot = TRUE, show_labels = FALSE,
          main='Euclidean-Complete')+
  geom_hline(yintercept = clHeight, linetype = 2, col="red")
```




```
pairs(iris4, col=cluster_h)                    # pairwise scatterplot with clusters
```



Divisive Hierarchical Clustering

It is a "top-down" approach, which generates a sequence of nested partitions of the data — progressively more granular:

- Start from one cluster containing all data points ($k = 1$)
- At each iteration:
  - Find the largest cluster (in terms of its diameter – i.e., the one with largest dissimilarity between any two of its observations)
  - Split it
  - Update the cluster list
- Iterate until all data points belong to a separate cluster (i.e., each point is its own cluster, so $k = n$)

To perform the Divisive Hierarchical Clustering we can use the **diana** function.

```
help(diana)
```

The function requires:

- ***x***: data matrix or data frame, or dissimilarity matrix

Using the previously computed `eu_iris` (i.e., the dissimilarity matrix for Iris data based on Euclidean distances):

```
hc_diana <- diana(eu_iris)
str(hc_diana)
```

```
## List of 6
##  $ order : int [1:150] 1 28 18 41 21 32 37 44 5 38 ...
##  $ height: num [1:149] 0.133 0.297 0.133 0.618 0.286 ...
##  $ dc    : num 0.94
##  $ merge : int [1:149, 1:2] -102 -8 -11 -10 -129 -18 -128 -3 -1 -81 ...
##  $ diss  : NULL
##  $ call  : language diana(x = eu_iris)
##  - attr(*, "class")= chr [1:2] "diana" "twins"
```

```
class(hc_diana)       # notice its difference from class(hc_centroid) etc.
```

```
## [1] "diana" "twins"
```

```
head(hc_diana$merge)
```

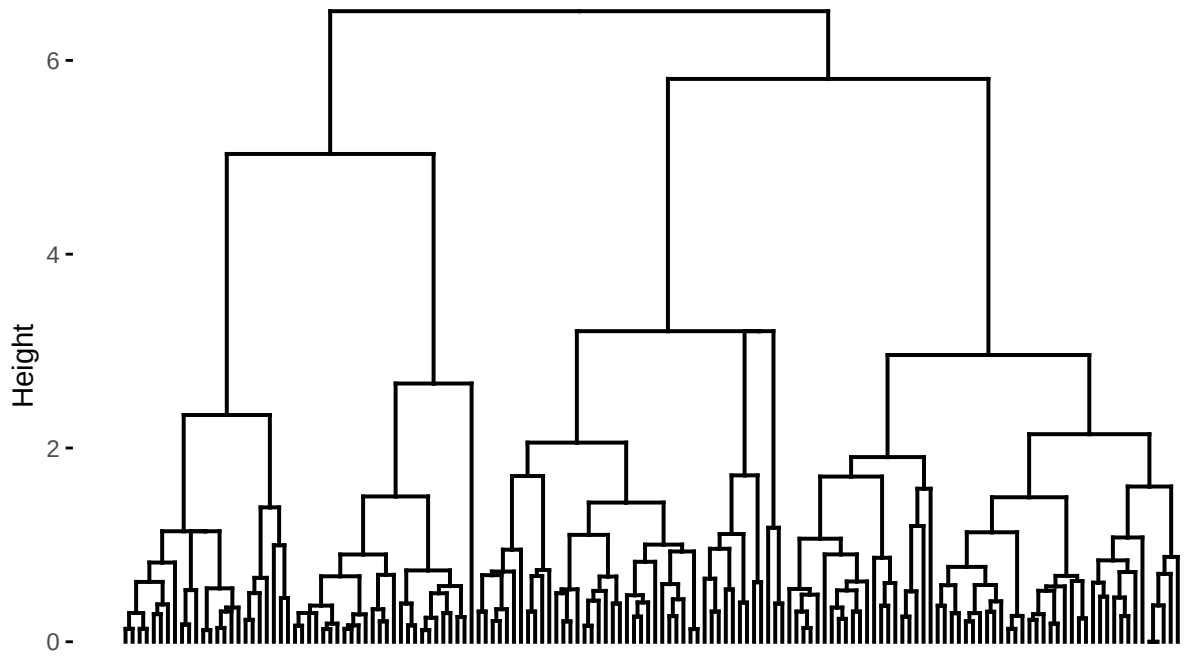
```
##      [,1] [,2]
## [1,] -102 -143
## [2,]   -8  -40
## [3,]  -11  -49
## [4,]  -10  -35
## [5,] -129 -133
## [6,]  -18  -41
```

```
hc_centroid
```

```
##
## Call:
## hclust(d = eu_iris, method = "centroid")
##
## Cluster method   : centroid
## Distance         : euclidean
## Number of objects: 150
```

To plot and cut the dendrogram we can use

```
fviz_dend(hc_diana, as.ggplot = TRUE,
          show_labels = FALSE, main='Euclidean-Complete') # plot the dendrogram
```



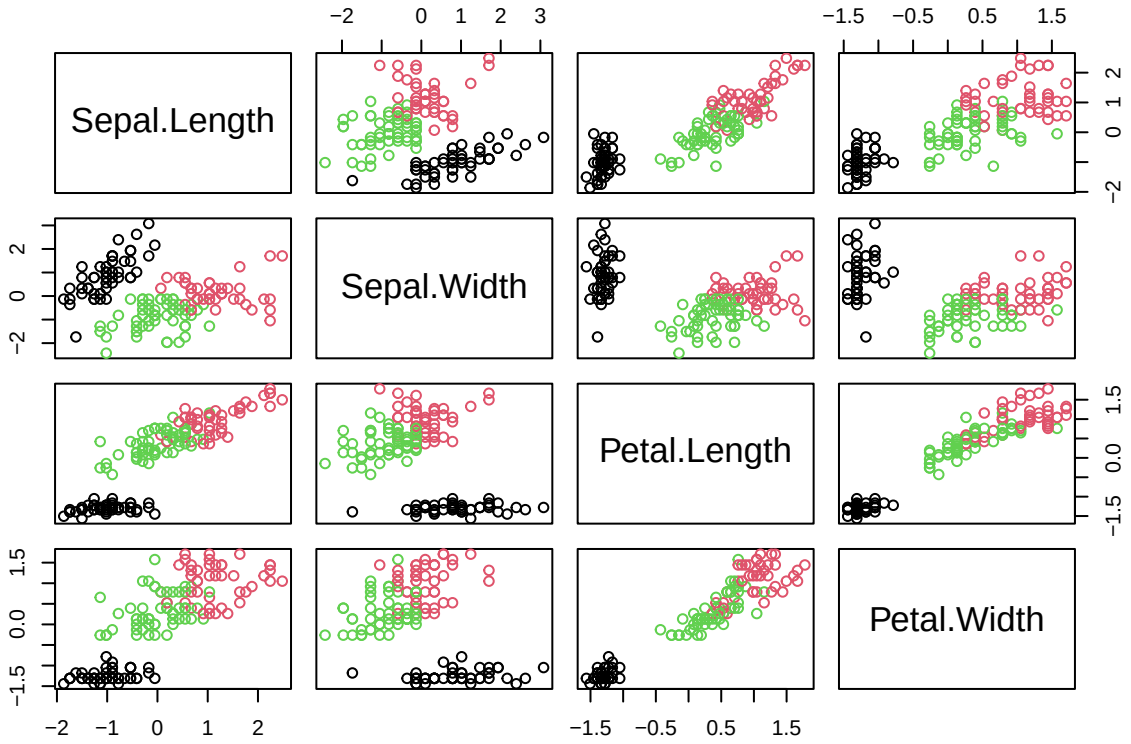

```
  cluster_diana <- cutree(hc_diana, k=3)                    # cut by k (with height? Error)
  cluster_diana
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 2 3 3 3 3 3 3 3 2 3 3 3 3 2 3 3 3
##  [75] 3 2 2 2 3 3 3 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 2 2 2 2 3 2 3 2 2
## [112] 3 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 2 3 3 2 2 2 3 2 2 2 3 2 2 2 3 2
## [149] 2 3
```

```
  cluster_diana <- cutree(as.hclust(hc_diana), h=5.5)       # cut by height (NOTE: convert it)
  cluster_diana
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 2 3 3 3 3 3 3 3 2 3 3 3 3 2 3 3 3
##  [75] 3 2 2 2 3 3 3 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 2 2 2 2 3 2 3 2 2
## [112] 3 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 2 3 3 2 2 2 3 2 2 2 3 2 2 2 3 2
## [149] 2 3
```

```
  pairs(iris4, col=cluster_diana)                           # pairwise scatterplot with clusters
```



k-means Clustering

k-means clustering is a partitioning algorithm that splits the data into k clusters by iteratively computing centroids/moving data points until convergence. Because the K-means algorithm finds a local optimum, the results obtained depend on the initial random cluster assignment. To perform the k-means clustering we can use the function **kmeans**.

```
help(kmeans)
```

We can see that the function requires:

- $\boldsymbol{x}$: numeric matrix of data
- **centers**: either the number of clusters, say k, or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in x is chosen as the initial centres.
- **nstart**:initial cluster assignments

Note: If a value of **nstart** greater than one is used, then K-means clustering will be performed using multiple random assignments in the initialization step, and the **kmeans()** function will report only the best results.

Therefore, a large **nstart** avoids an undesirable local optimum!!

```
set.seed(86)
km.2.1<-kmeans(iris4, 3, nstart=1)
km.2.10<-kmeans(iris4, 3, nstart=10)
km.2.30<-kmeans(iris4, 3, nstart=30)
km.2.1$tot.withinss
```

```
## [1] 189.7512
```

```
km.2.10$tot.withinss
```

```
## [1] 138.8884
```

```
km.2.30$tot.withinss
```

```
## [1] 138.8884
```

Using the Iris data set **iris4**, for $k = 3$ and $nstart = 10$:

```
  res <- kmeans(iris4, 3, nstart=10)
  str(res)
```

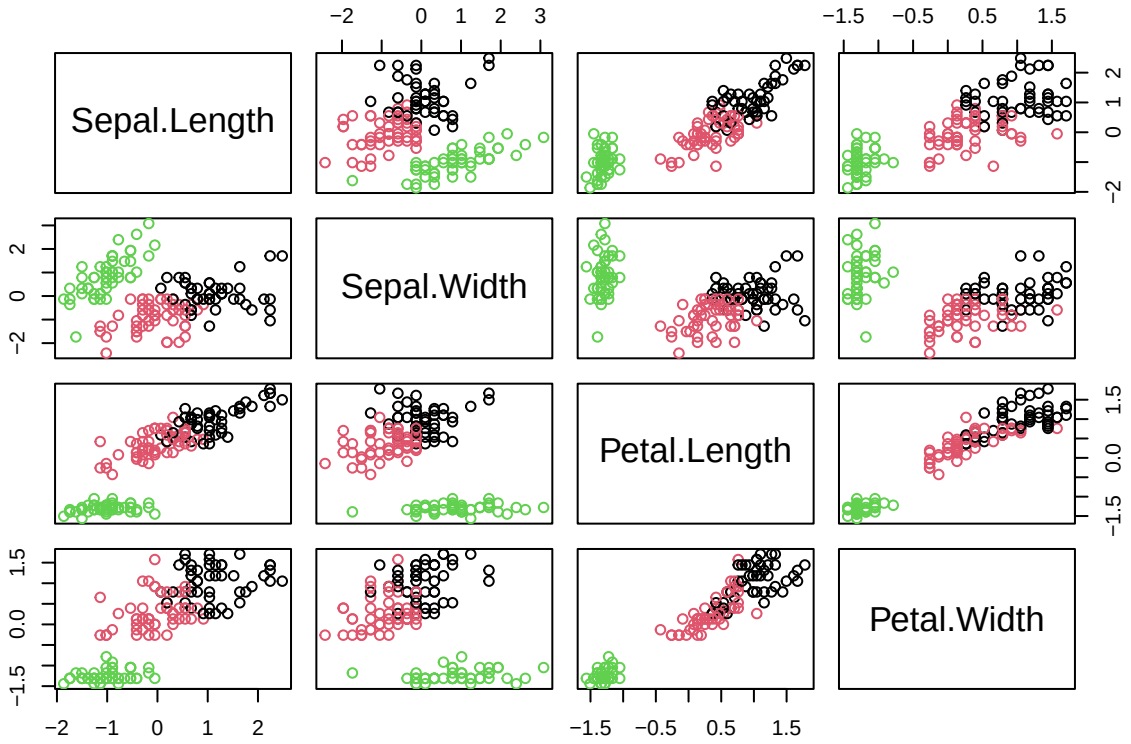
```
## List of 9
##  $ cluster     : int [1:150] 3 3 3 3 3 3 3 3 3 3 ...
##  $ centers     : num [1:3, 1:4] 1.1322 -0.0501 -1.0112 0.0881 -0.8804 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "1" "2" "3"
##   .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##  $ totss       : num 596
##  $ withinss    : num [1:3] 47.5 44.1 47.4
##  $ tot.withinss: num 139
##  $ betweenss   : num 457
##  $ size        : int [1:3] 47 53 50
##  $ iter        : int 3
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

The clusters are stored in `res$cluster`:

```
  res$cluster
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2
##  [75] 2 1 1 1 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1
## [112] 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 1 1 1 1 1 1 2 2 1 1 1 2 1 1 1 2 1 1 1 2 1
## [149] 1 2
```

```
  pairs(iris4, col=res$cluster) # pairwise scatterplot with clusters
```



Hierarchical Clustering and k-means with one single command

Using the function **eclust** (in **factoextra**) it is possible to perform both methods. Its advantages include:

- It requires a single function call (instead of using two different command, packages, etc.)
- Automatically computes the "gap statistics" to estimate the number of clusters
- It provides silhouette information for all partitioning methods and hierarchical clustering
- It draws beautiful and sexy graphs using **ggplot2**

Let's compute Agglomerative Hierarchical Clustering using **eclust**.

```
hc_res <- eclust(iris4,                          # data
                 "hclust",                       # method
                 k = 3,                          # num. of clusters
                 hc_metric = "euclidean",        # distance measure
                 hc_method = "single")           # linkage function
str(hc_res)
```

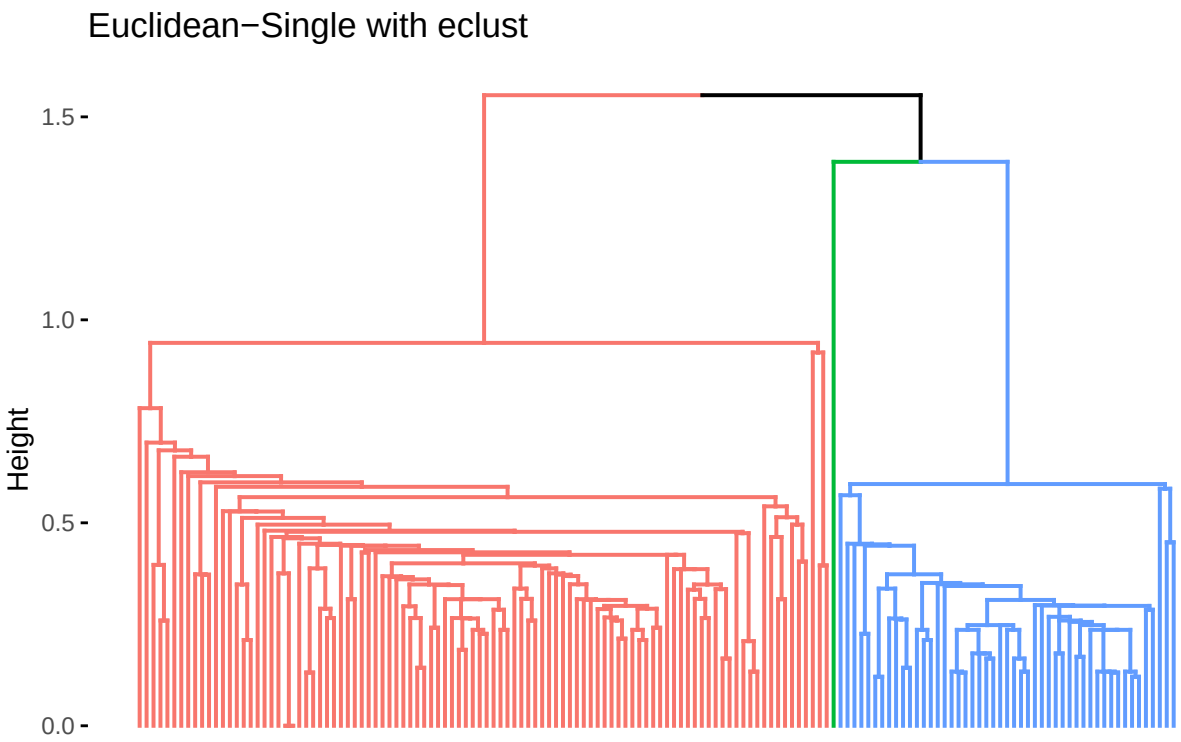
```
## List of 12
##  $ merge      : int [1:149, 1:2] -102 -8 -11 -10 -1 -129 -41 -128 -28 -3 ...
##  $ height     : num [1:149] 0 0.121 0.121 0.131 0.131 ...
##  $ order      : int [1:150] 107 61 99 58 94 109 63 119 88 69 ...
##  $ labels     : NULL
##  $ method     : chr "single"
##  $ call       : language stats::hclust(d = x, method = hc_method)
```

```
##  $ dist.method: chr "euclidean"
##  $ cluster    : int [1:150] 1 1 1 1 1 1 1 1 1 1 ...
##  $ nbclust    : num 3
##  $ silinfo    :List of 3
##   ..$ widths         :'data.frame':  150 obs. of  3 variables:
##   .. ..$ cluster  : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ neighbor : num [1:150] 2 2 2 2 2 2 2 2 2 2 ...
##   .. ..$ sil_width: num [1:150] 0.715 0.715 0.713 0.711 0.71 ...
##   ..$ clus.avg.widths: num [1:3] 0.56 0 0.483
##   ..$ avg.width      : num 0.505
##  $ size       : int [1:3] 49 1 100
##  $ data       : num [1:150, 1:4] -0.898 -1.139 -1.381 -1.501 -1.018 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##   ..- attr(*, "scaled:center")= Named num [1:4] 5.84 3.06 3.76 1.2
##   .. ..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##   ..- attr(*, "scaled:scale")= Named num [1:4] 0.828 0.436 1.765 0.762
##   .. ..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##  - attr(*, "class")= chr [1:3] "hclust" "hcut" "eclust"
```

```
hc_res$cluster
```

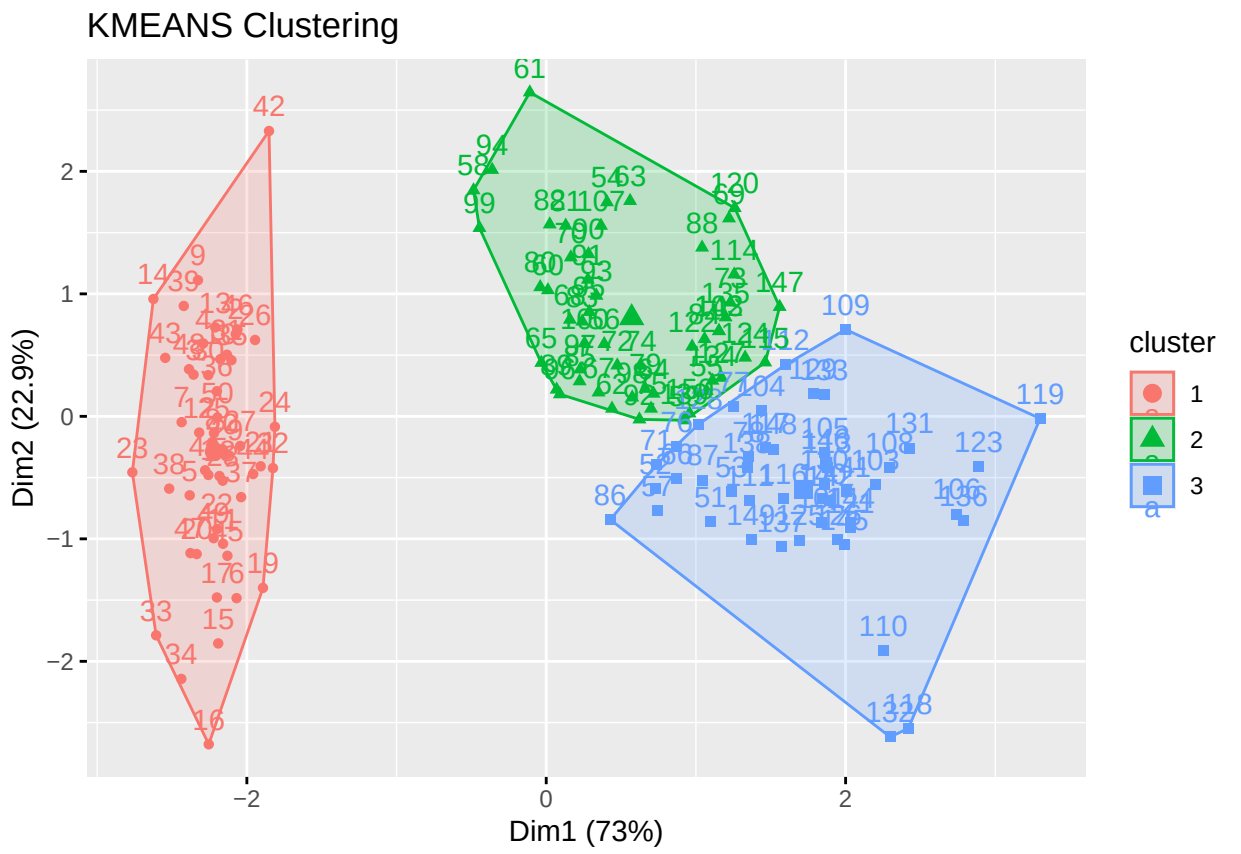
```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 2 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

```
fviz_dend(hc_res, as.ggplot = TRUE,
          show_labels = FALSE,
          main='Euclidean-Single with eclust') # plot the dendrogram
```



Let's compute k-means using eclust. You can notice that the clusters are represented in a 2D scatterplot based on the first two Principal Components (NOTE: wait/see the next lesson).

```
# it receives data, algorithm, k, distance to compute
km_res <- eclust(iris4, "kmeans", k = 3, hc_metric = "euclidean")
```




```
km_res$cluster
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 2 2 3 2 2 2 2 2 2 2 3 2 2 2 2 3 2 2 2
##  [75] 2 3 3 3 2 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 3 3 3 3 3 3 2 2 3 3 3 2 3 3 3 2 3 3 3 2 3
## [149] 3 2
```

Evaluating a Clustering Solution

Besides dendrogram cut by height (shorter cut means smaller and more compact clusters), or final value of the total within cluster sum of squares (**tot.withinss** for k-means), a clustering can be evaluated through *Silhouette widths*.

We can use the **silhouette** command.

```
help(silhouette)
```

We can see that the function requires:

- $\boldsymbol{x}$: an integer vector with k different integer cluster codes (with $2 \leq k \leq n-1$)
- **dist**: a dissimilarity object

It returns *"an object, sil, of class silhouette which is an $n \times 3$ matrix with attributes. For each observation i, $sil[i,]$ contains the cluster to which i belongs as well as the neighbor cluster of i (the cluster, not containing i, for which the average dissimilarity between its observations and i is minimal), and the silhouette width $s(i)$ of the observation."*

Remark: points with a large $s(i)$ (approaching 1) are very well clustered. The ones with small $s(i)$ (close to 0) lie between two clusters. Observations with negative $s(i)$ (approaching -1) are probably placed in the wrong cluster.

Let's run the **silhouette** command based on k-means partition.

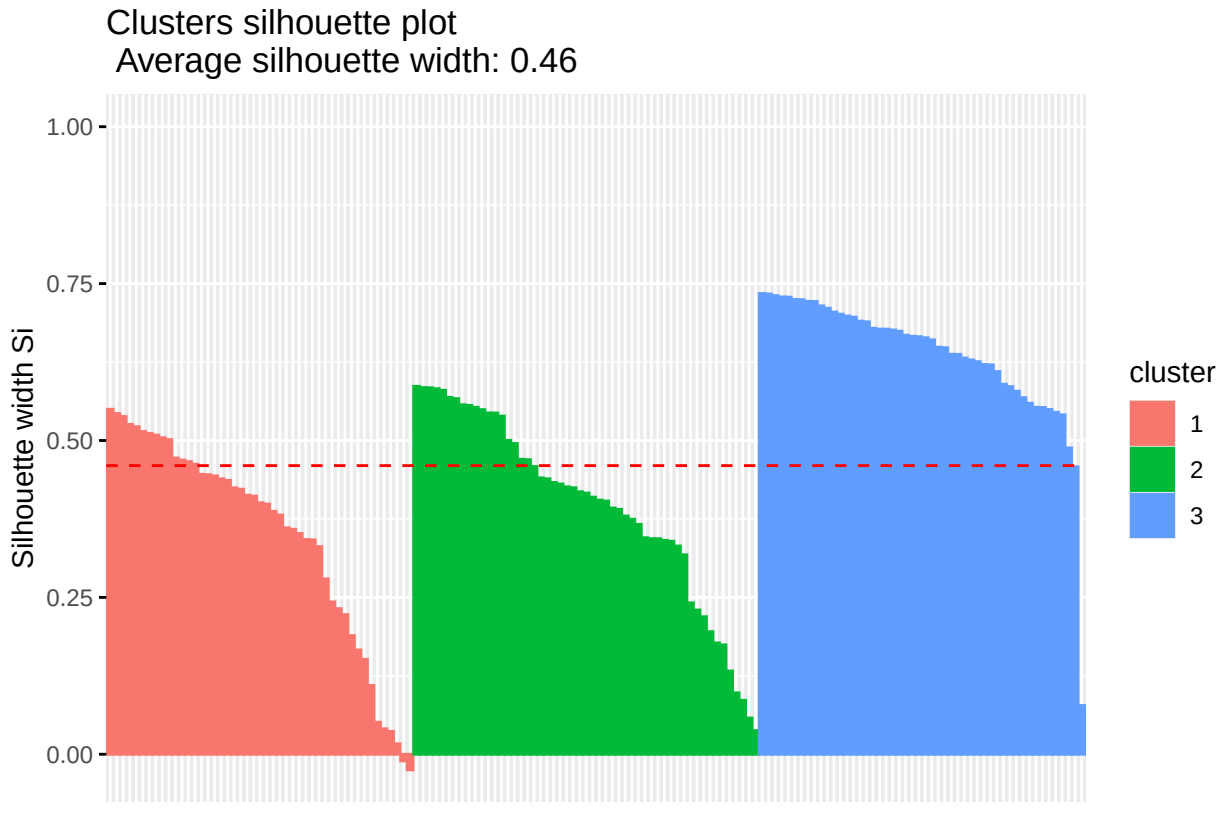
```
distance <- dist(iris4, method="euclidean")
sil <- silhouette(x = res$cluster, dist = distance)
sil[1:5,] # showing the first 5 results
```

```
##      cluster neighbor sil_width
## [1,]       3        2 0.7341949
## [2,]       3        2 0.5682739
## [3,]       3        2 0.6775472
## [4,]       3        2 0.6205016
## [5,]       3        2 0.7284741
```

To get a Silhouette plot we will use the **factoextra** environment.

```
fviz_silhouette(sil)
```

```
##   cluster size ave.sil.width
## 1       1   47          0.35
## 2       2   53          0.39
## 3       3   50          0.64
```



Approaches to determine the number of clusters in a data set

We can determine the number of clusters in a data set using different strategies:

- Within cluster dissimilarity/distance (**tot.withinss**)
- Hartigan Index
- Average Silhouette

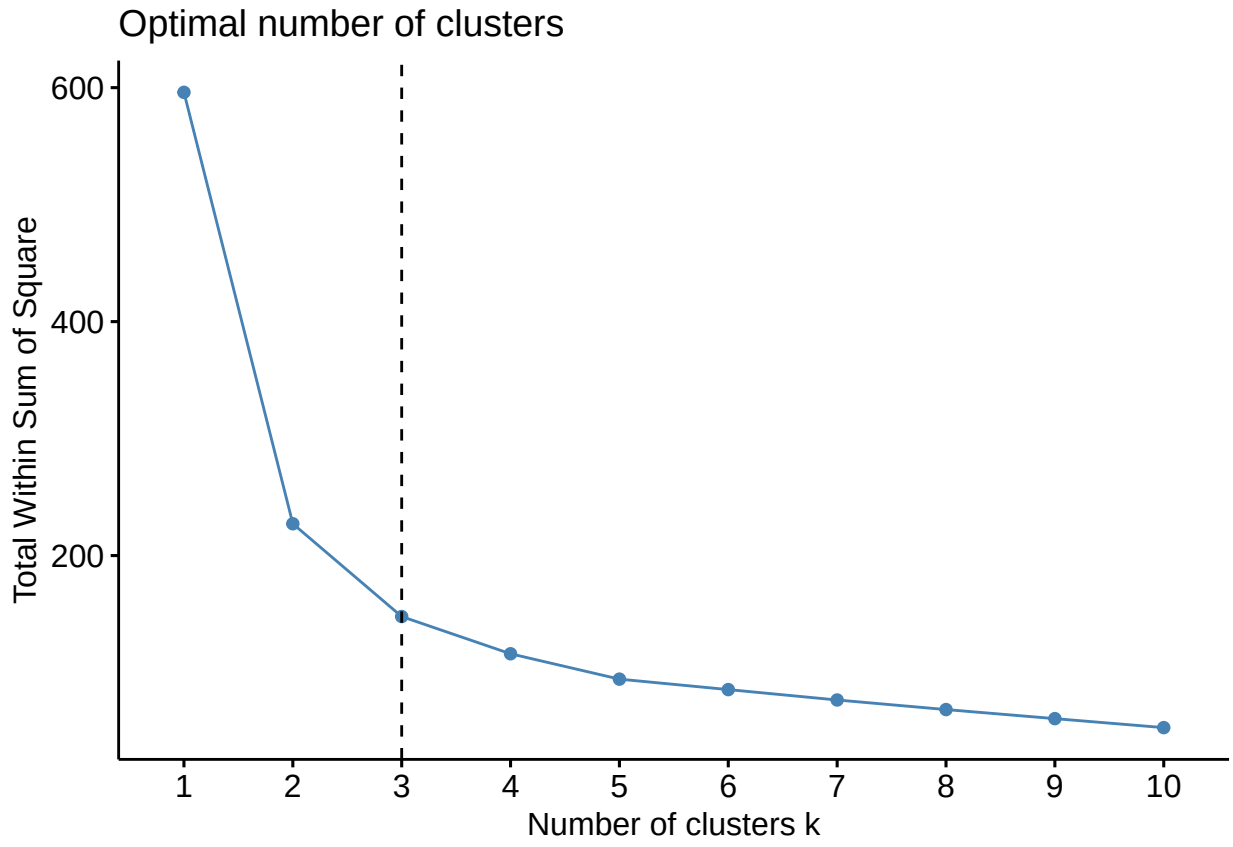
Within cluster dissimilarity/distance

- **Hierarchical:** Dissimilarity levels (heights) at which clusters are formed
- ***k*-means:** Within clusters sum of squares (it is guaranteed to be a local minimum for any given random initialization)

We can use **fviz-nbclust**, and use the *elbow method* (look at the knee).

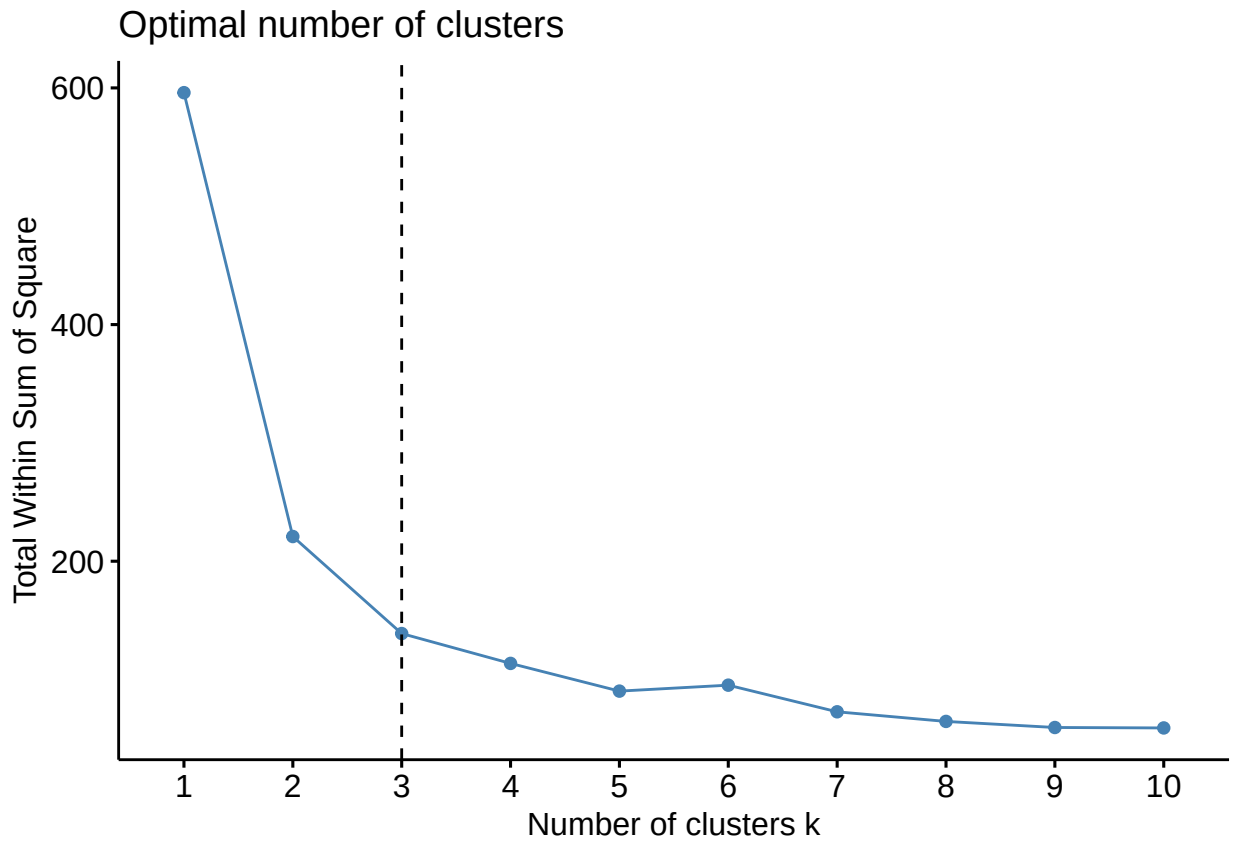
For the Agglomerative Hierarchical Clustering:

```
fviz_nbclust(iris4, hcut, method = "wss") +
  geom_vline(xintercept = 3, linetype = 2)
```



For the *k*-means

```
fviz_nbclust(iris4, kmeans, method = "wss") +
  geom_vline(xintercept = 3, linetype = 2)
```

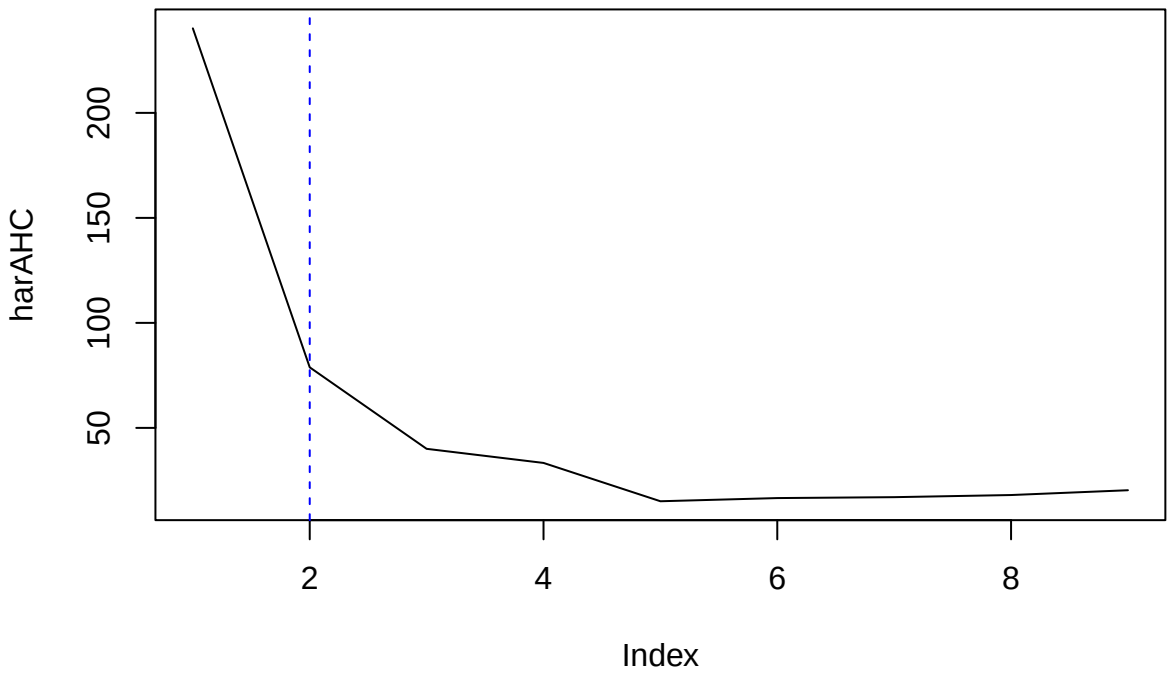


Hartigan Index

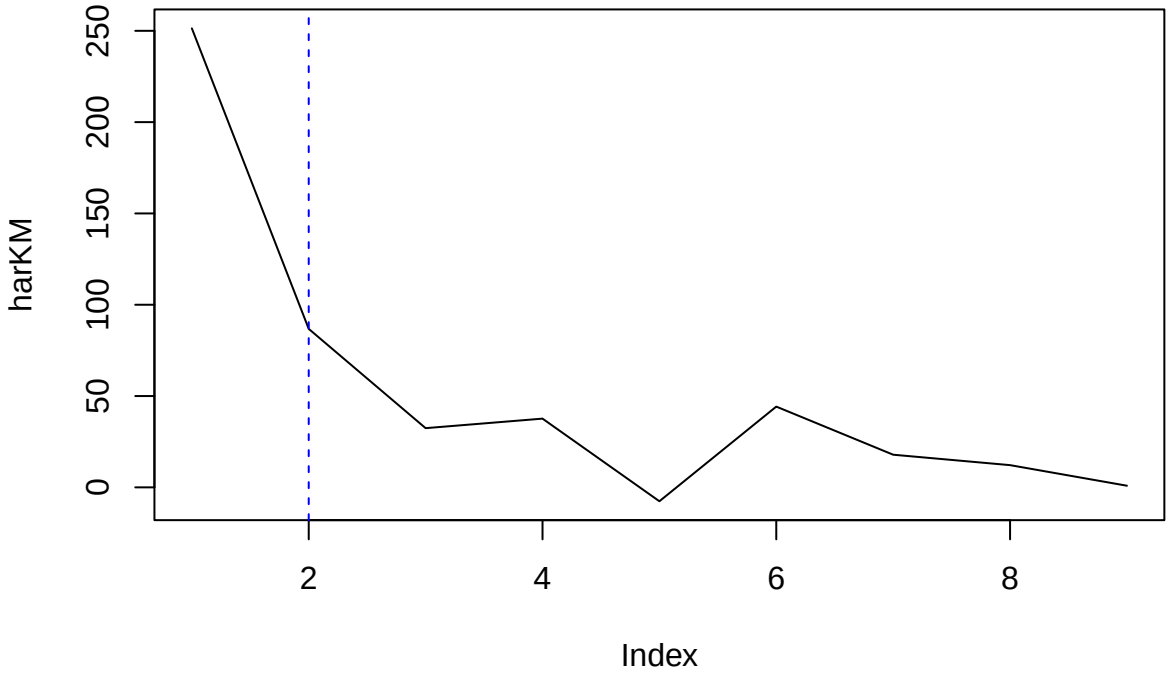
It measures the relative change of fitness as the number of clusters changes. *Remark*: only the data we want to cluster are needed.

We can use the wss calculated previously

```
WSS_hc <-fviz_nbclust(iris4, hcut, method = "wss")$data[,2]
harAHC <- NULL
for (i in 1:(length(WSS_hc)-1)) {
  harAHC[i] <- (nrow(iris4)-i-1)*(WSS_hc[i]-WSS_hc[i+1])/WSS_hc[i+1]
  Best.nc_hc<-which.max(harAHC)+1
}
plot(harAHC,type="l")
abline(v=Best.nc_hc, col="blue", lty=2)
```



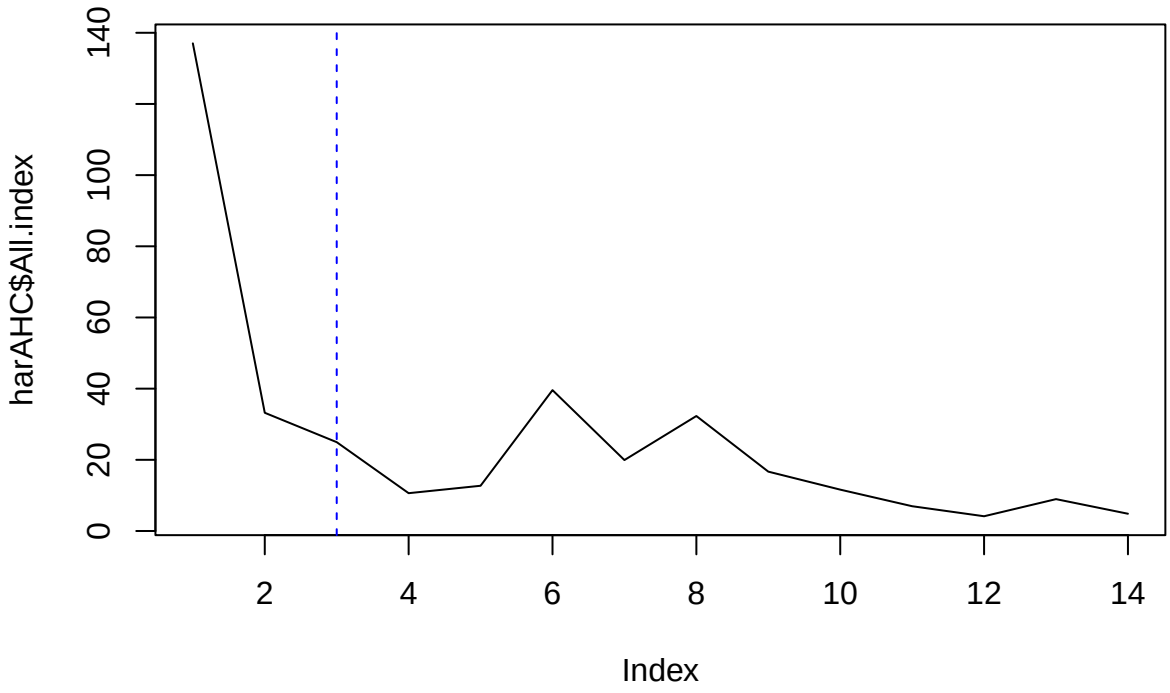

```
WSS_km<-fviz_nbclust(iris4, kmeans, method = "wss")$data[,2]
harKM <- NULL
for (i in 1:(length(WSS_km)-1)) {
  harKM[i] <- (nrow(iris4)-i-1)*(WSS_km[i]-WSS_km[i+1])/WSS_km[i+1]
  Best.nc_km<-which.max(harKM)+1
}
plot(harKM,type="l")
abline(v=Best.nc_km, col="blue", lty=2)
```



Alternatively, we can use the function **NbClust** (from the **NbClust** package).

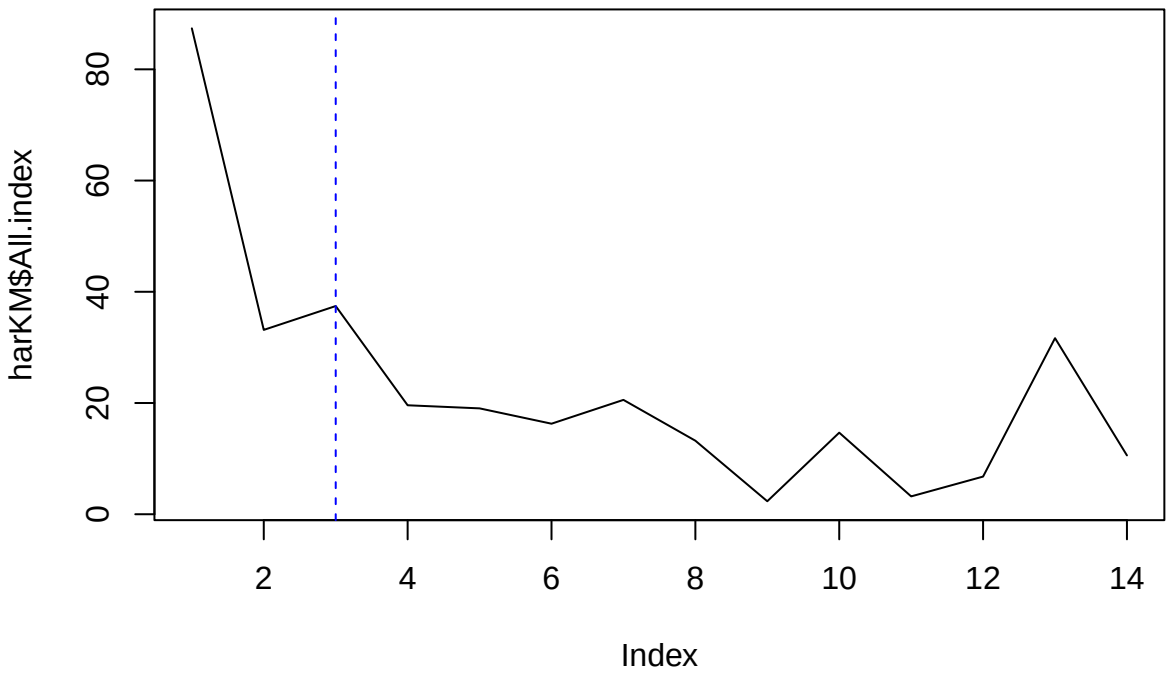
For Agglomerative Hierarchical Clustering:

```
harAHC <- NbClust(iris4,  distance = "euclidean", method = "complete", index='hartigan')
plot(harAHC$All.index, type = "l")
abline(v=harAHC$Best.nc[1], col="blue", lty=2)
```



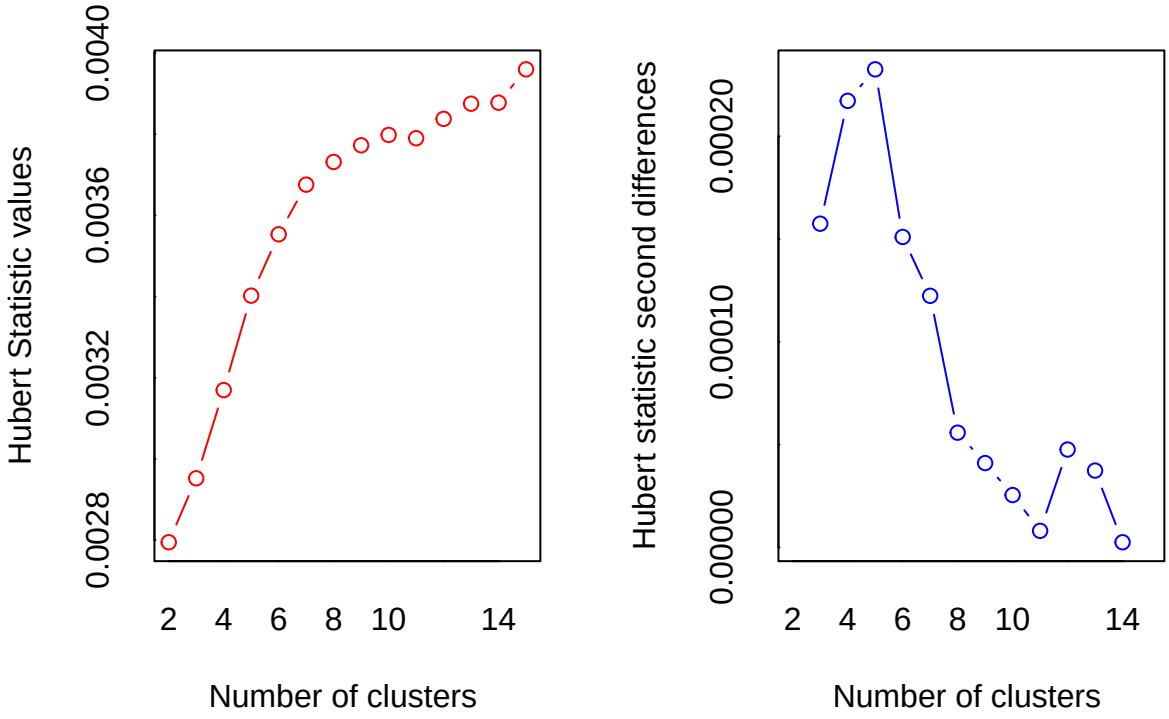
For k-means:

```
harKM <- NbClust(iris4,  distance = "euclidean", method = "kmeans", index='hartigan')
plot(harKM$All.index, type = "l")
abline(v=harKM$Best.nc[1], col="blue", lty=2)
```

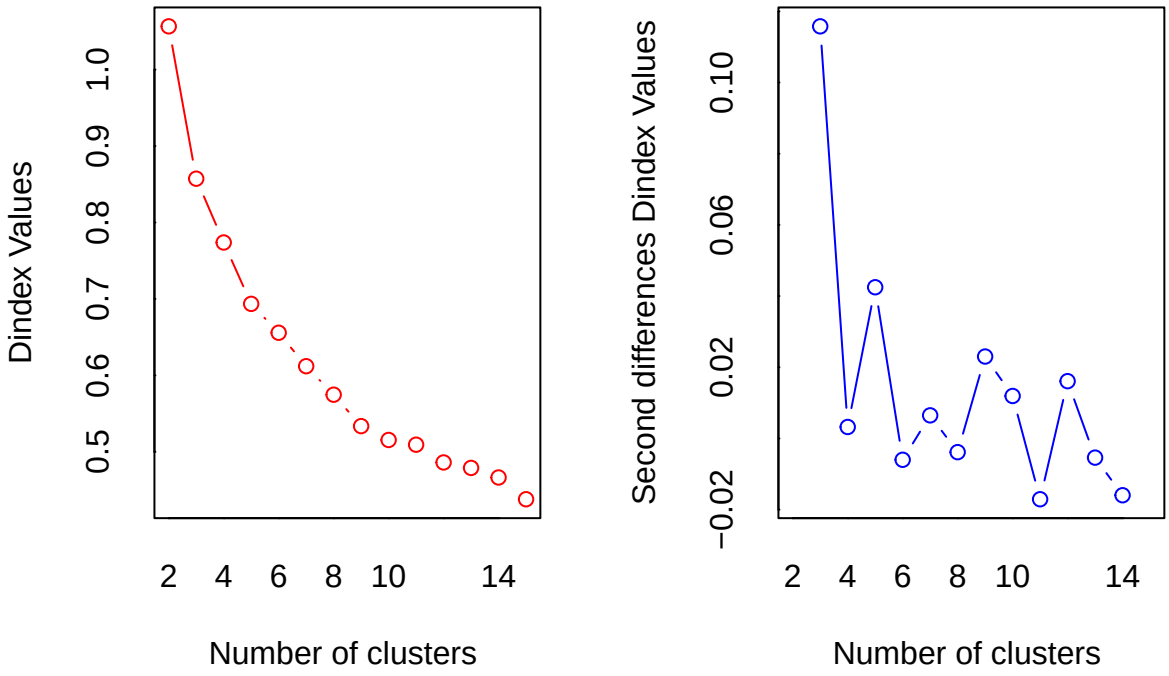


Remark: using **index='all'**, the same function provides 30 indices for determining the number of clusters and proposes to user the best clustering scheme from the different results obtained by varying all combinations of number of clusters. The same holds for distance measures, clustering methods, etc.

```
help(NbClust)
allKM <- NbClust(iris4,  distance = "euclidean", method = "kmeans", index='all')
```




```
## *** : 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.
##
```



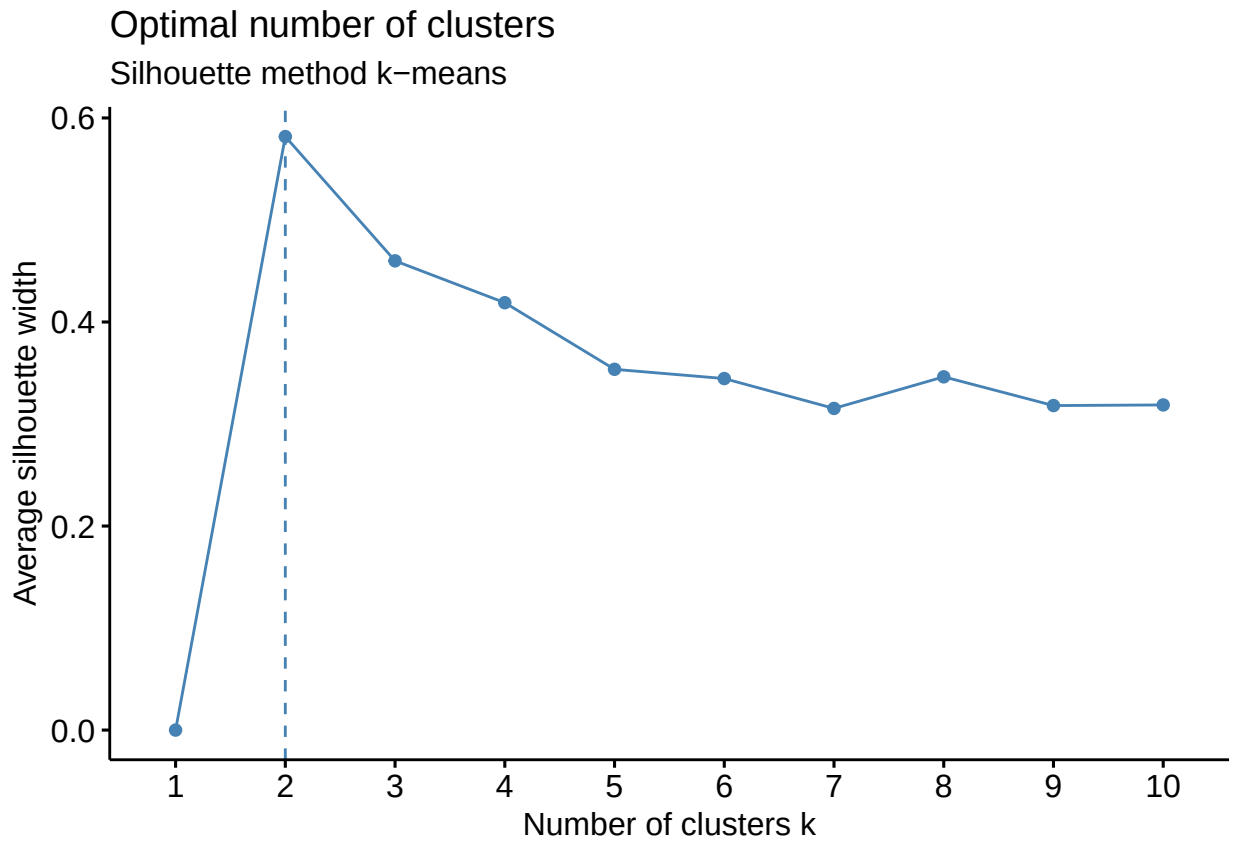

```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 10 proposed 2 as the best number of clusters
## * 6 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 3 proposed 12 as the best number of clusters
## * 1 proposed 14 as the best number of clusters
## * 2 proposed 15 as the best number of clusters
##
##                   ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

Average Silhouette

We can use **fviz-nbclust**.

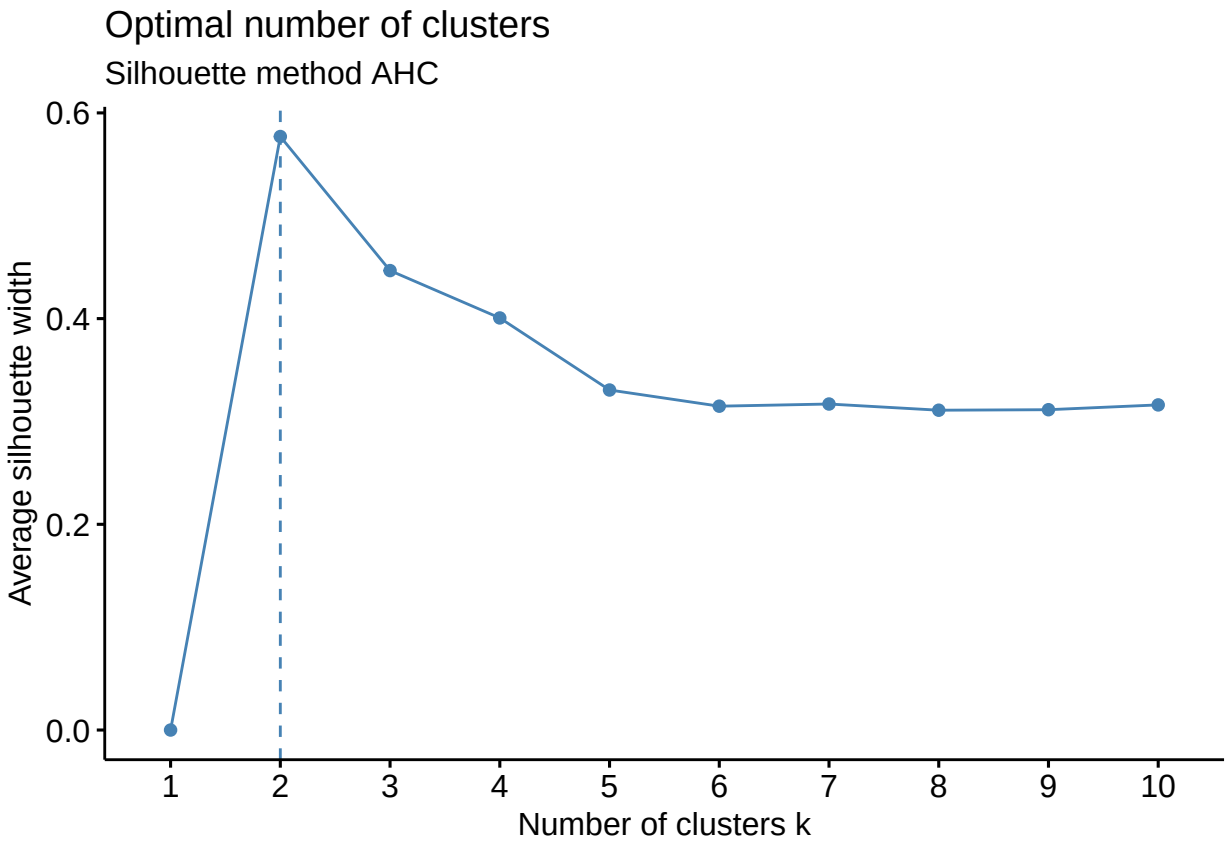
For k-means:

```
# Silhouette method
fviz_nbclust(iris4, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method k-means")
```



For Agglomerative Hierarchical Clustering:

```
# Silhouette method
fviz_nbclust(iris4, hcut, method = "silhouette")+
  labs(subtitle = "Silhouette method AHC")
```



Clustering: Further topics

Model-based clustering based on finite Gaussian mixture models can be performed using the **Mclust** library through the **Mclust** command. Here models are estimated by an EM algorithm initialized by hierarchical model-based agglomerative clustering. The optimal model is then selected according to a Bayesian Information Criterion.

The command **adjustedRandIndex** computing the *adjusted Rand Index* is in the same library.

Another library that can be used is **clusterR**.

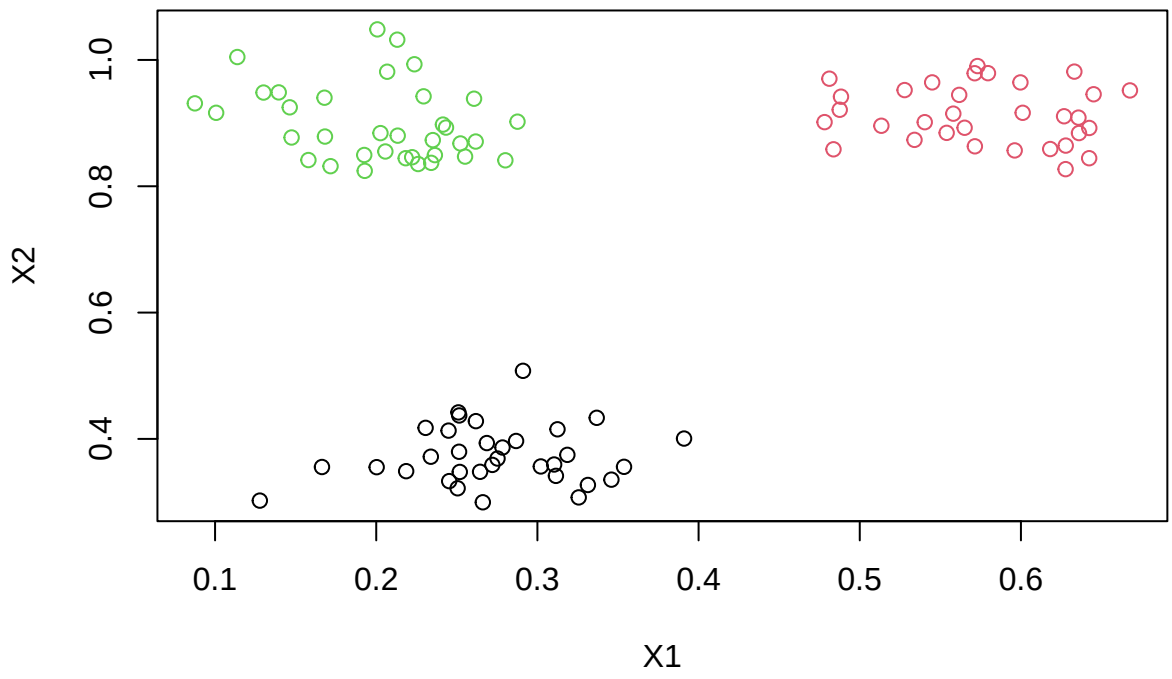
Generating GMM

Generating Gaussian mixtures is non-trivial and requires a certain number of choices, such as the (average and maximum) overlap across groups, the sphericity (diagonal vs non-diagonal covariance structure), homogeneity (equal covariance across groups), etc.

Fortunately, we can use the **MixSim** package for this task.

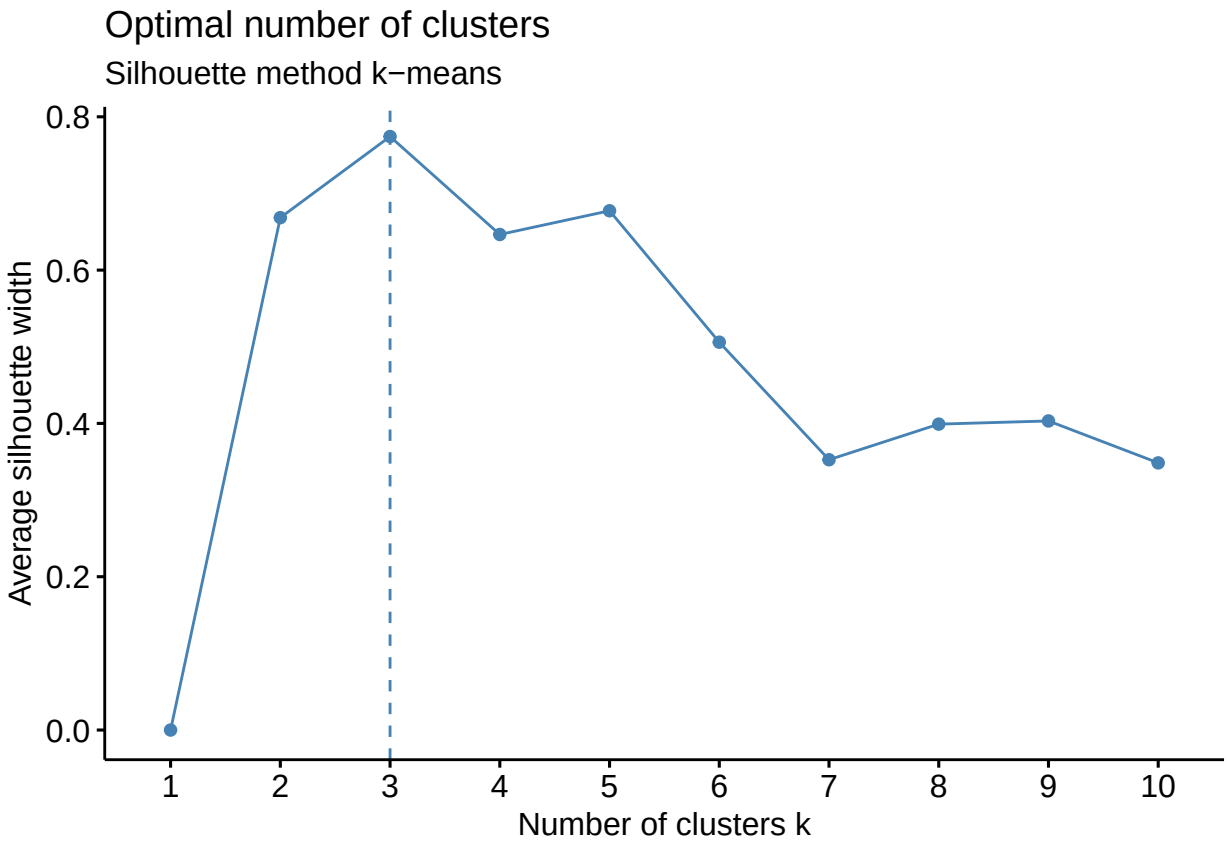
```
set.seed(1)        # fix the results (there is plenty of randomness)

require(MixSim)    # laod the package
help(MixSim)

k <- 3             # num. of groups
p <- 2             # dimensionality
hom <- T           # equal covariances
```

```r
sph <- T           # spherical components
MaxOmega <- 0.001 # maximum degree of overlap

# generate the finite mixture model with Gaussian components
A <- MixSim(MaxOmega = MaxOmega, K = k, p = p, sph = sph, hom=hom)

# simulate the dataset
n <- 100           # sample size
alpha <- 0.01      # outliers
x <- simdataset(n, A$Pi, A$Mu, A$S, n.noise = 0, n.out = 0, alpha = alpha)

# rename stuff
colnames(x$X) = c("X1", "X2")
id = x$id          # unit labels
x = x$X            # n times p data points
plot(x, col=id)
```

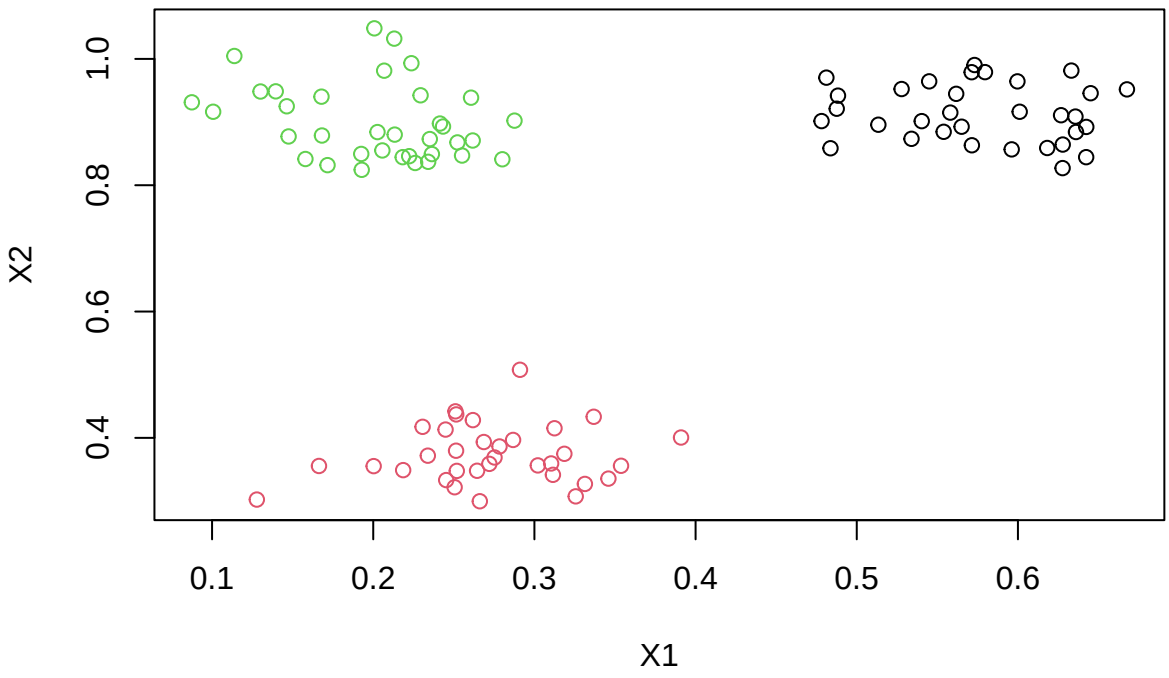


Here k-means assumptions are fully satisfied and does a great job.

```r
# Silhouette method
kmsol <- fviz_nbclust(x, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method k-means")
plot(kmsol)
```



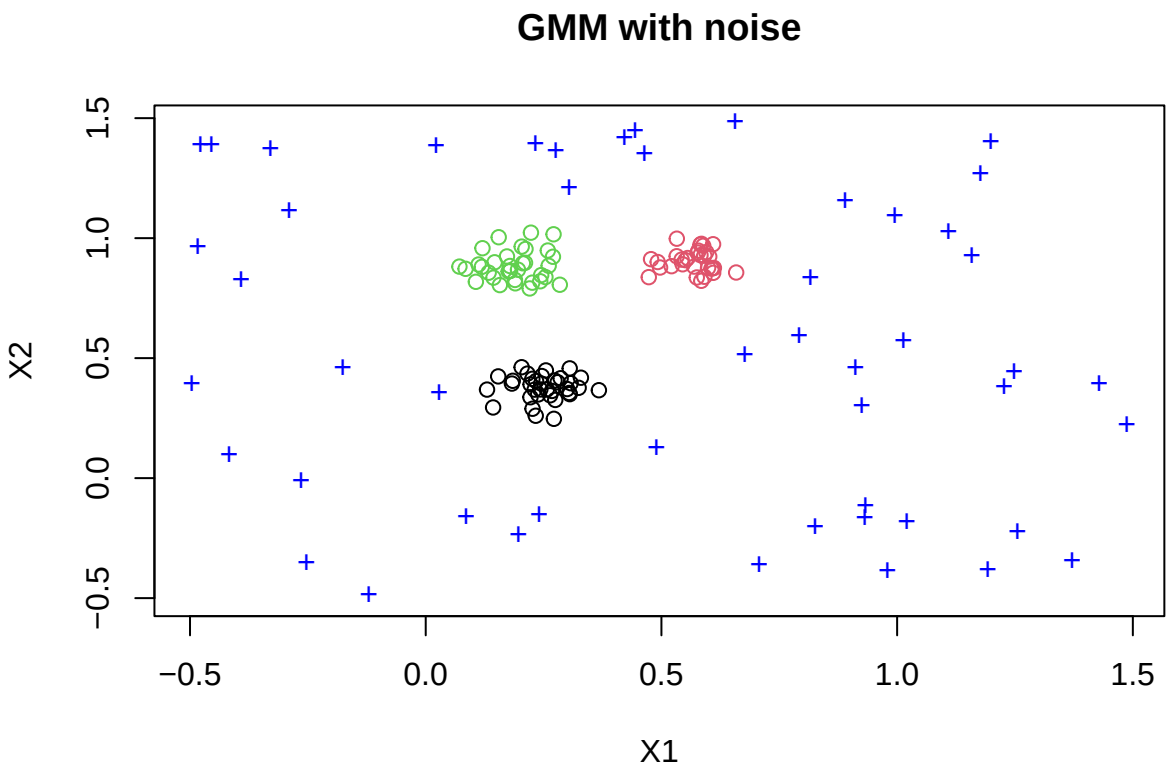

```
# plot the solution
res <- kmeans(x, 3, nstart=10)
plot(x, col=res$cluster) # pairwise scatterplot colored in clusters
```



Contaminated data

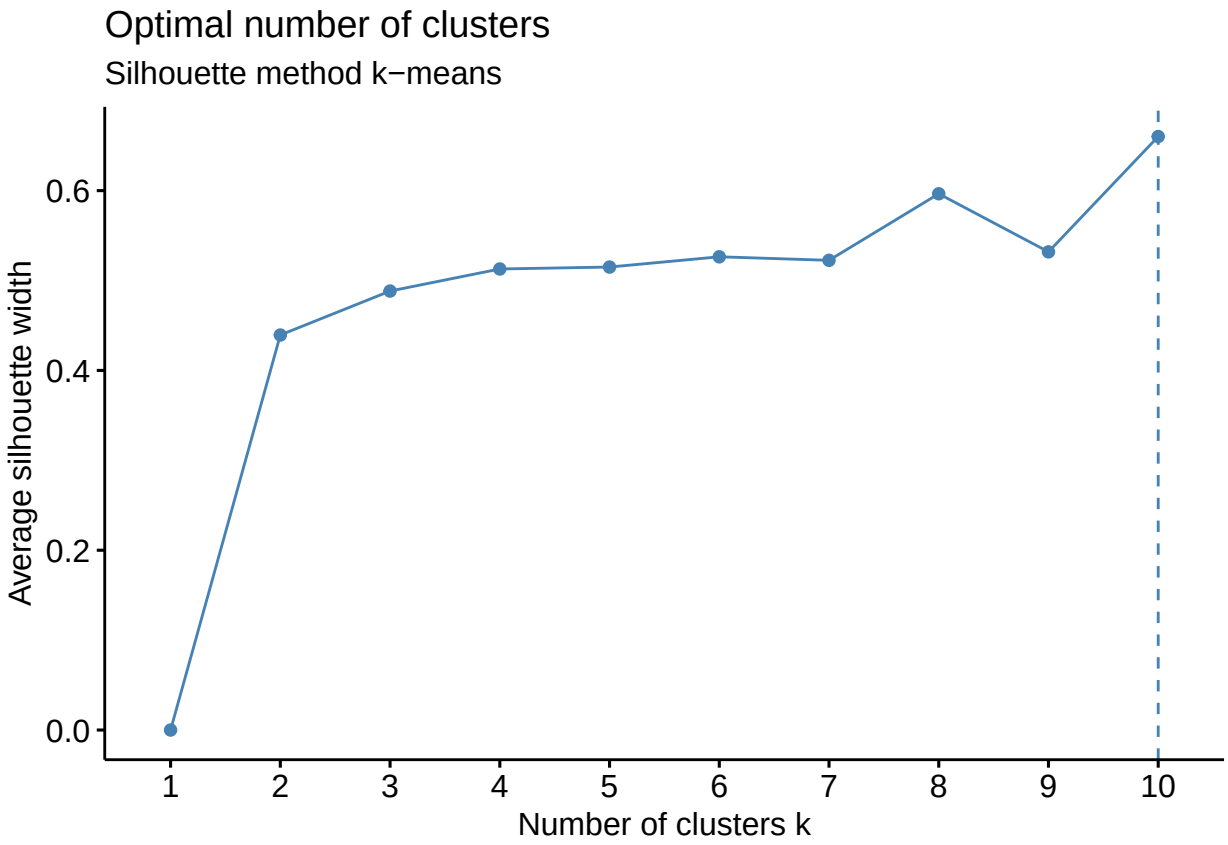
What if the data contain some spurious observations? Let's see what happens with some additional random noise.

```
set.seed(1)

# simulate the dataset
n <- 100        # sample size
numout <- 50    # num of outliers
x <- simdataset(n, A$Pi, A$Mu, A$S, n.noise = 0,
                n.out=numout, int=c(-0.5, 1.5))

colnames(x$X) = c("X1", "X2")
id = x$id       # unit labels
x = x$X         # n times p data points
plot(x, col=id, main="GMM with noise")
points(x[id==0, ], pch="+", col = "blue")
```

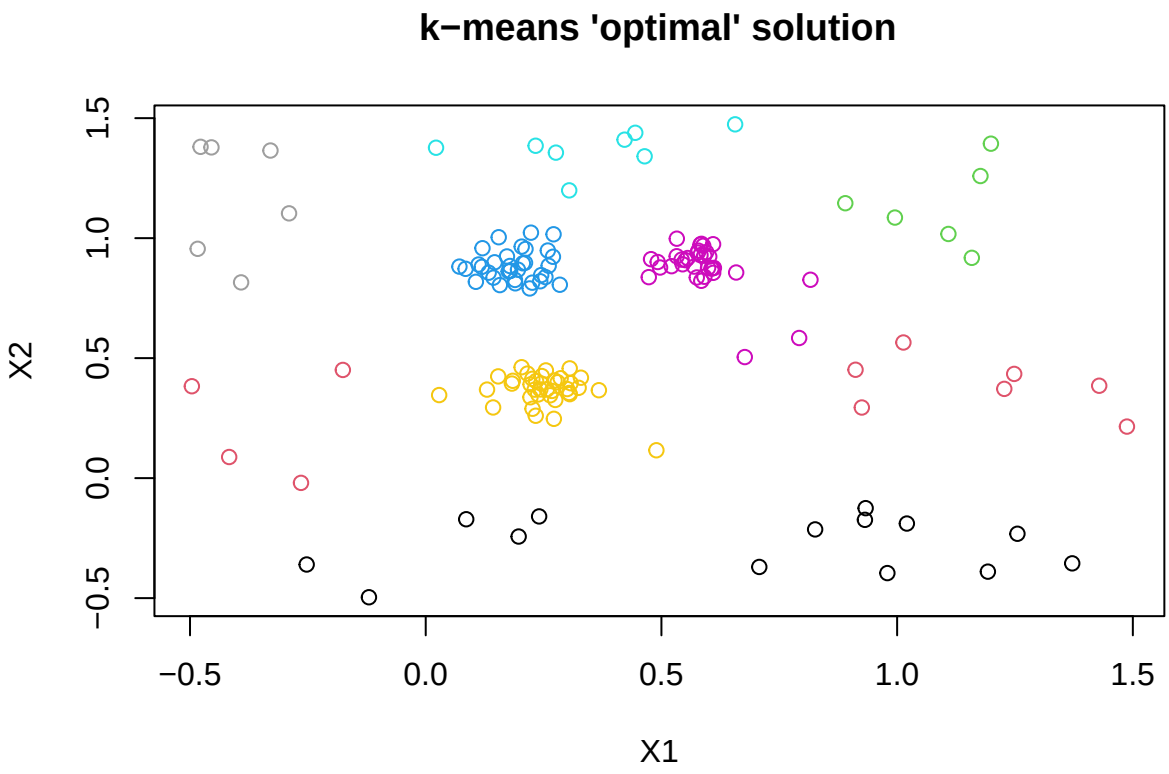


Noisy data break down k-means, and we see that the estimated number of groups is largely inflated.

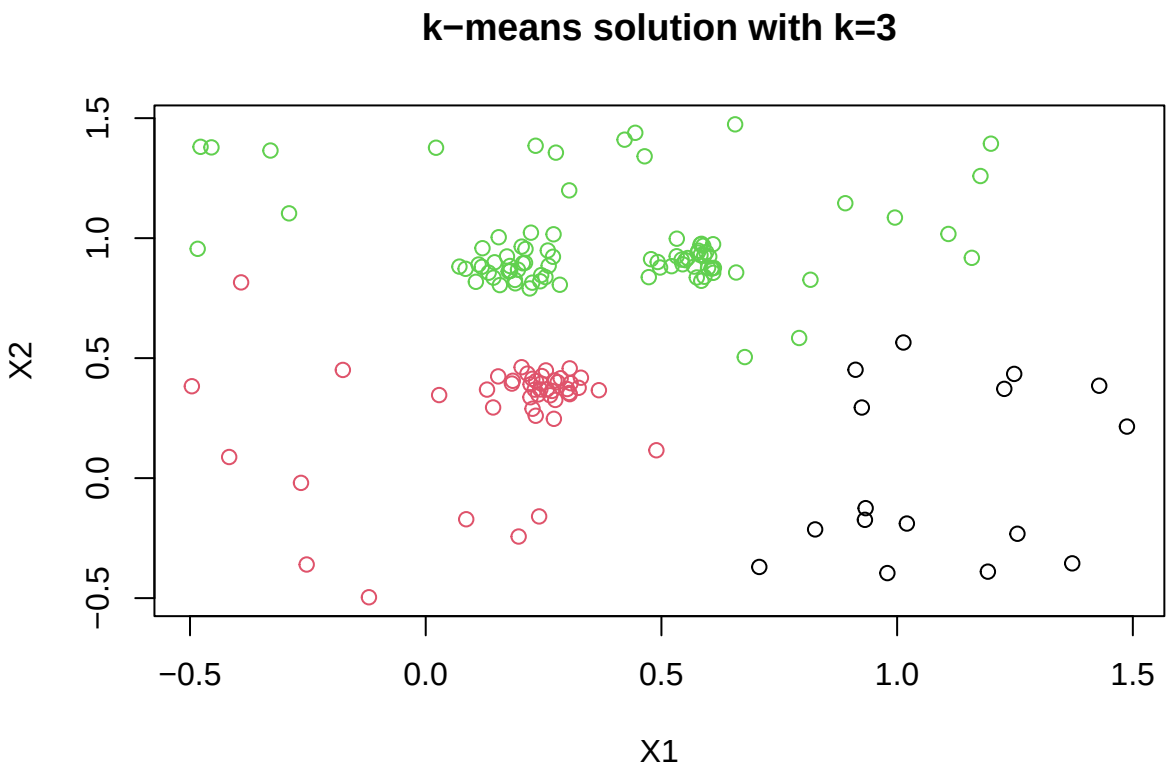
```
# Silhouette method
kmsol <- fviz_nbclust(x, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method k-means")
plot(kmsol)
```




```
# plot the "optimal" solution
res <- kmeans(x, 10, nstart=10)
plot(x, col=res$cluster,
     main="k-means 'optimal' solution") # pairwise scatterplot with clusters
```




```
# original estimate of k
res <- kmeans(x, 3, nstart=10)
plot(x, col=res$cluster,
     main="k-means solution with k=3") # pairwise scatterplot with clusters
```

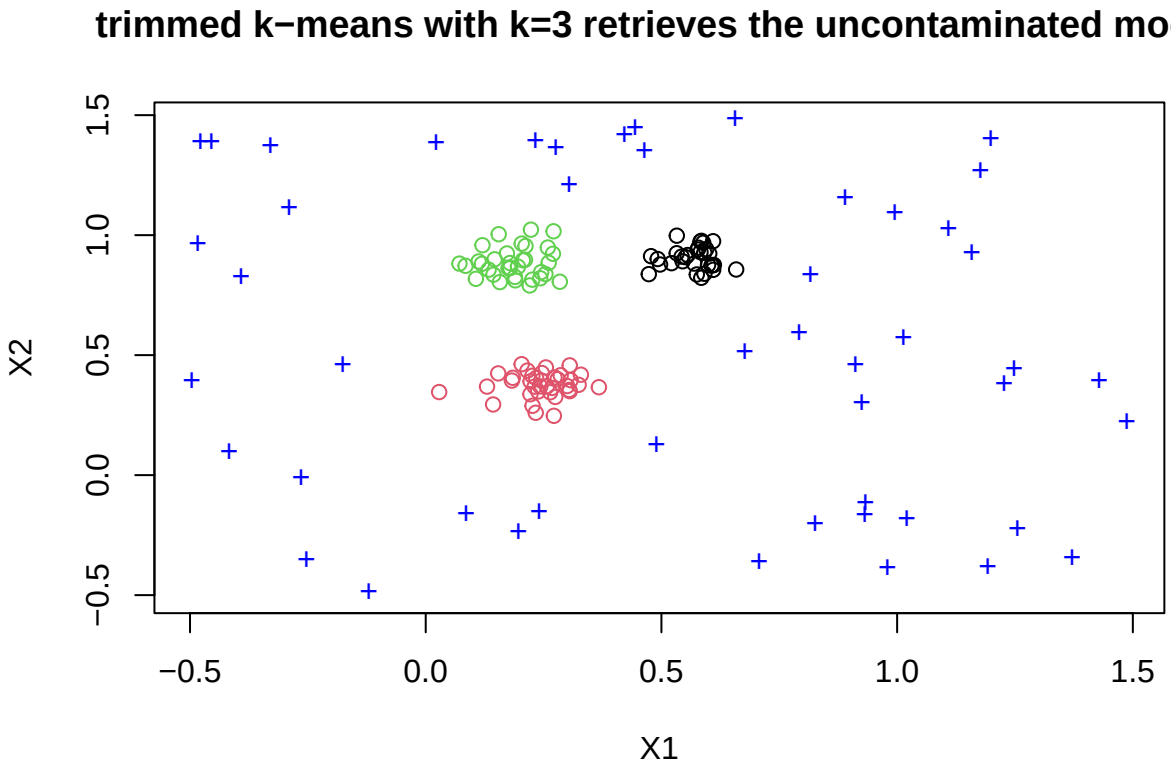


Remark: Noisy observations can be modeled as part of the mixture; **Mclust** can do this.

However, let's try a more "agnostic" approach. *Trimmed k-means* (available through the package **trimcluster**) is a "robust" counterpart of k-means. It excludes a fraction h/n of points that contribute the most to its objective function (i.e. the most aberrant points)and so encompasses k-means as a special case (i.e. no trimming for $h = 0$).

Remark: It is a combinatorial optimization problem, but this package uses an heuristic algorithm that guarantees to achieve a local optimum.

```
# install.packages('trimcluster')
library('trimcluster')
k = 3                              # num. of groups
trimProp = numout/(n+numout)       # trimming propotion (i.e., h/n)
tkm <- trimkmeans(x, k=k, trim=trimProp)

# assign the estimated labels for outliers and non-outlying cases
out <- tkm$classification == k+1
nonout <- tkm$classification != k+1

# plot the solution
plot(x[nonout, ], col = tkm$classification[nonout],
     main="trimmed k-means with k=3 retrieves the uncontaminated model",
     xlim = c(min(x[,1]), max(x[,1])),
     ylim = c(min(x[,2]), max(x[,2])))
points(x[out, ], col = "blue", pch="+")
```




```
# Note: also here clustering evaluation can be performed in several ways
# (i.e., we do not need to know that k=3 in this example)
```

options(tinytex.verbose = TRUE)

Extension to elliptical components: **tclust** generalizes Mclust through an additional trimming proportion. Unlike trimmed k-means, this is very effective also in the presence of non-homogeneous and/or non-spherical components with unequal number of points. Check the **tclust** package if interested.