Clustering

J. Di Iorio, F. Chiaromonte

2/19/2021

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).

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**,**factoextra** and **NbClust**

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

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

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

```
library(NbClust)
```

Data

Today we are going to use the ICONIC **Anderson's Iris data set** (https://en.wikipedia.org/wiki/Iris_flower_data_set) without the *Species* label. The data set consists of 50 samples from each of three species of Iris (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

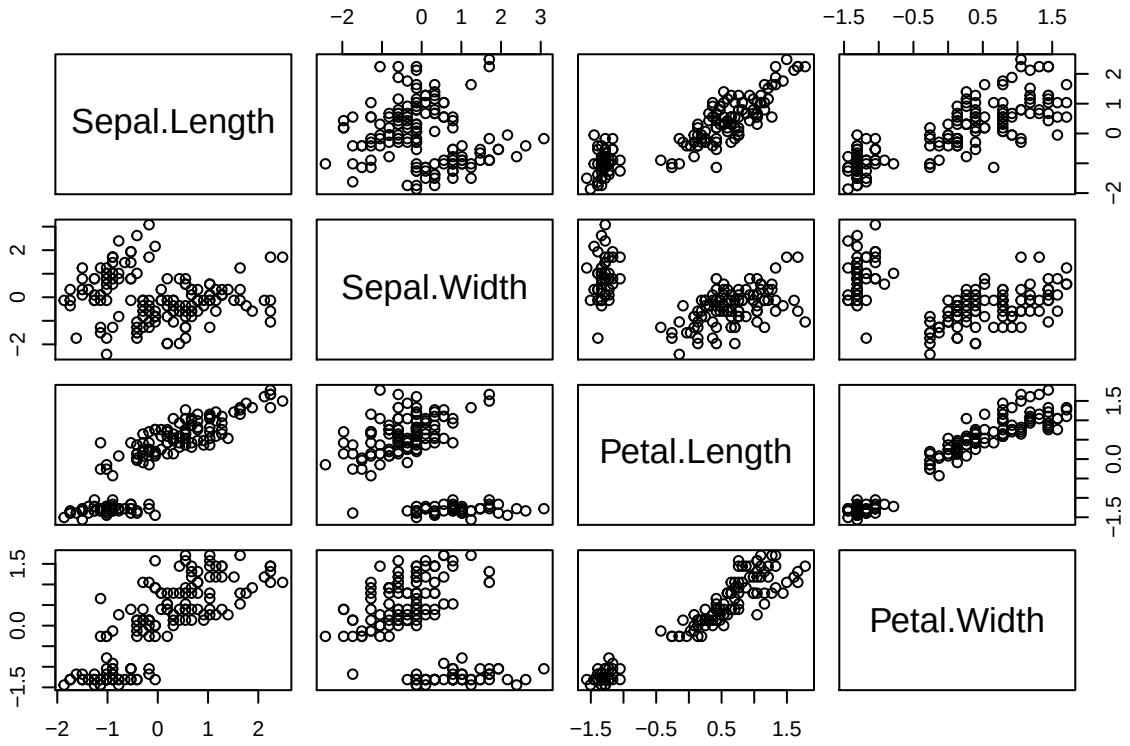
The Iris dataset is already available in the **cluster** package.

```
library(cluster)
help(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 do you think there are?

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



Hierarchical Clustering

These algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances. The hierarchy is usually represented by a dendrogram. In a dendrogram, the y-axis marks the distance at which the clusters merge, while the objects are placed along the x-axis such that the clusters don't mix.

Strategies for hierarchical clustering generally fall into two types:

- Agglomerative Hierarchical Clustering
- Divisive Hierarchical Clustering

Agglomerative Hierarchical Clustering

Proceeding in an agglomerative fashion ("bottom-up"), it generates a sequence of nested partitions of the data – progressively less fine:

- Start from n clusters, each containing one data point
- At each iteration:
  - Find the two closest clusters.
  - Merge them and update the list of clusters.

  - Update the matrix of cluster distances.
- Iterate until all data points belong to the same cluster

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 structure based on **Euclidean distance** can be produced by **dist** in the following way:

```
eu_iris <- dist(iris4, method='euclidean')
```

Now we are ready to create a hierarchy of data trying different linkage methods.

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

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

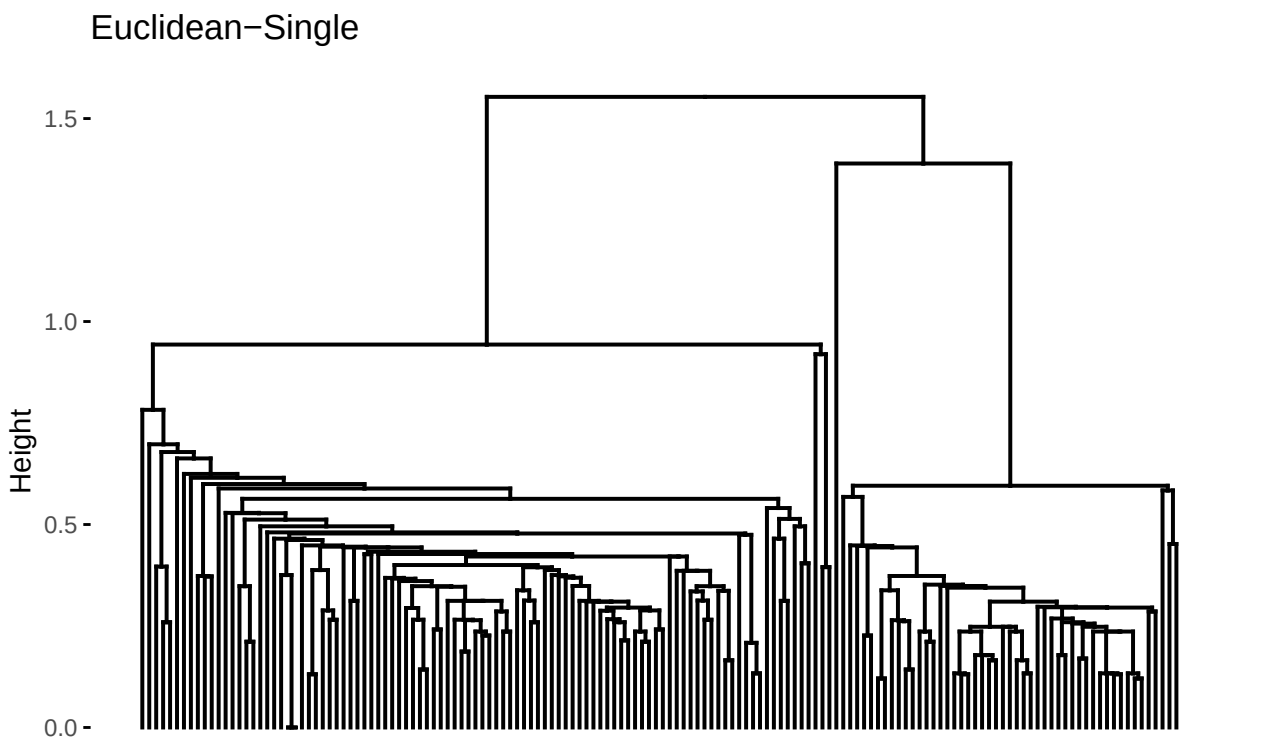
```
## List of 7
##  $ merge      : int [1:149, 1:2] -102 -8 -11 -10 -1 -129 -41 -128 -28 -3 ...
##  $ height     : num [1:149] 0 0.121 0.121 0.131 0.131 ...
##  $ order      : int [1:150] 107 61 99 58 94 109 63 119 88 69 ...
##  $ labels     : NULL
##  $ method     : chr "single"
##  $ call       :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

```
head(hc_single$merge) # steps
```

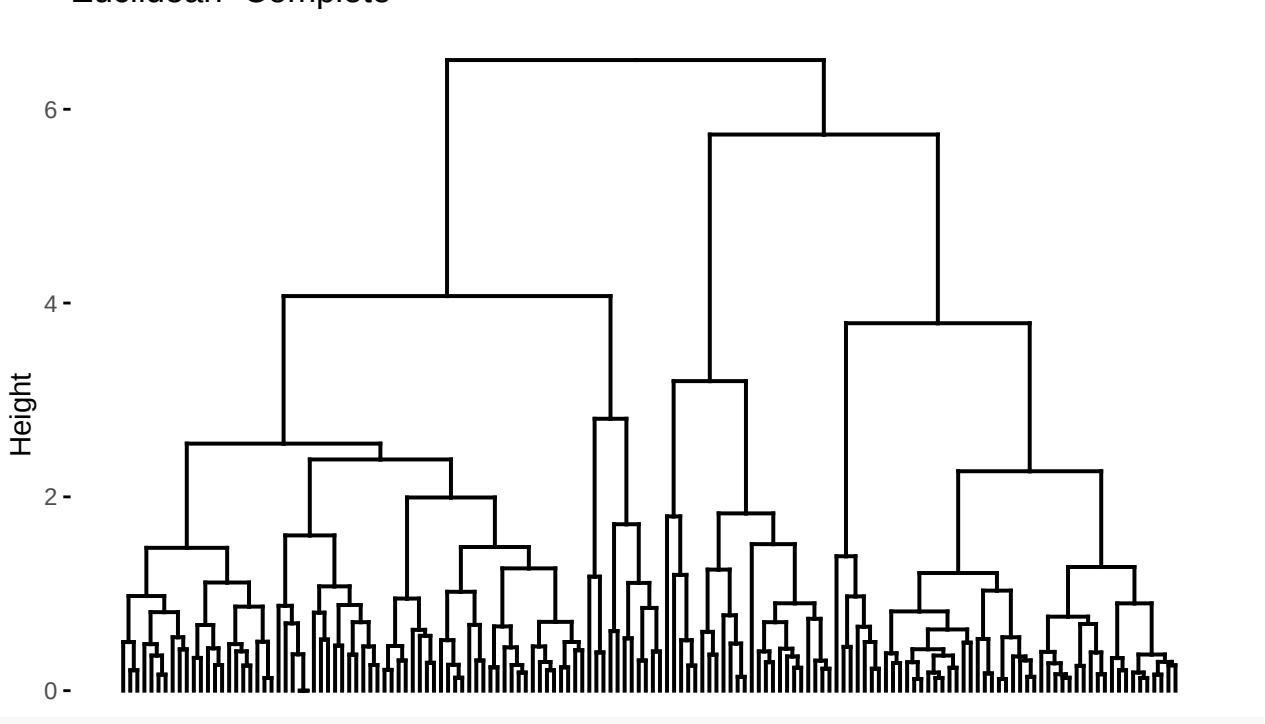
```
##      [,1] [,2]
## [1,] -102 -143
## [2,]   -8  -40
## [3,]  -11  -49
## [4,]  -10  -35
## [5,]   -1  -18
## [6,] -129 -133
```

The hierarchies are represented using dendrograms: a plot illustrating the arrangement of the clusters produced.

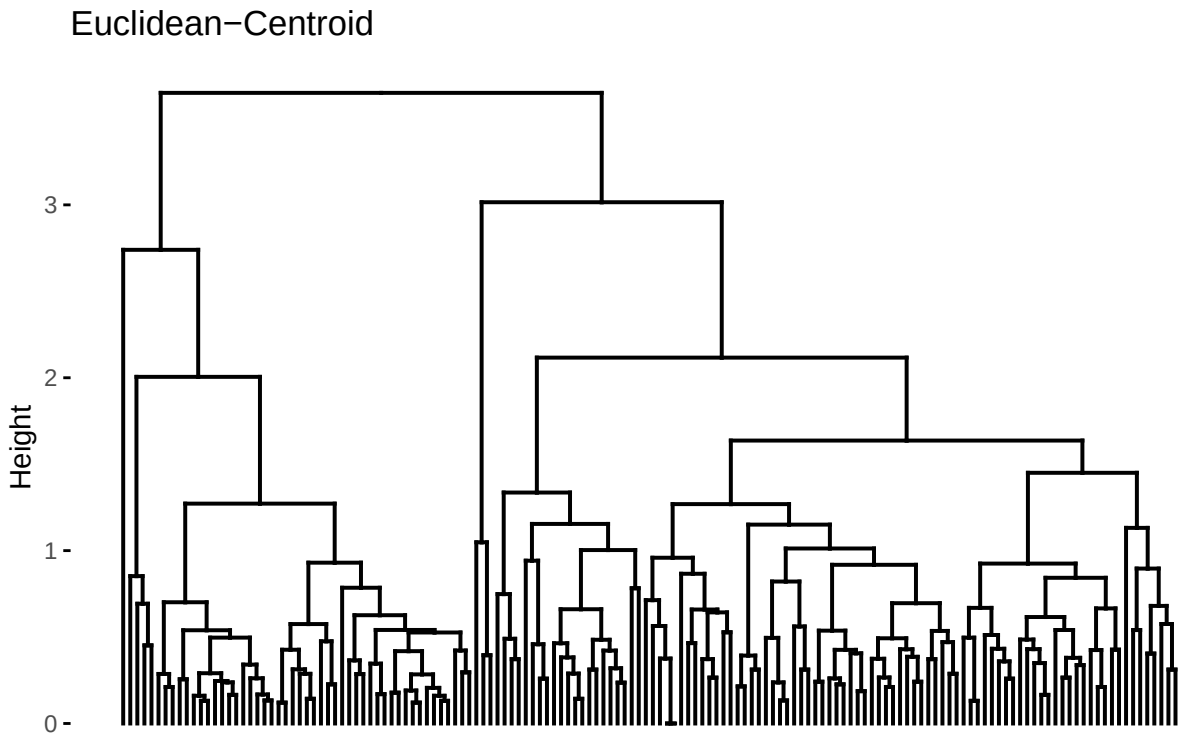
```
par(mfrow=c(2,2))
fviz_dend(hc_single, as.ggplot = TRUE, show_labels = FALSE, main='Euclidean-Single')
```



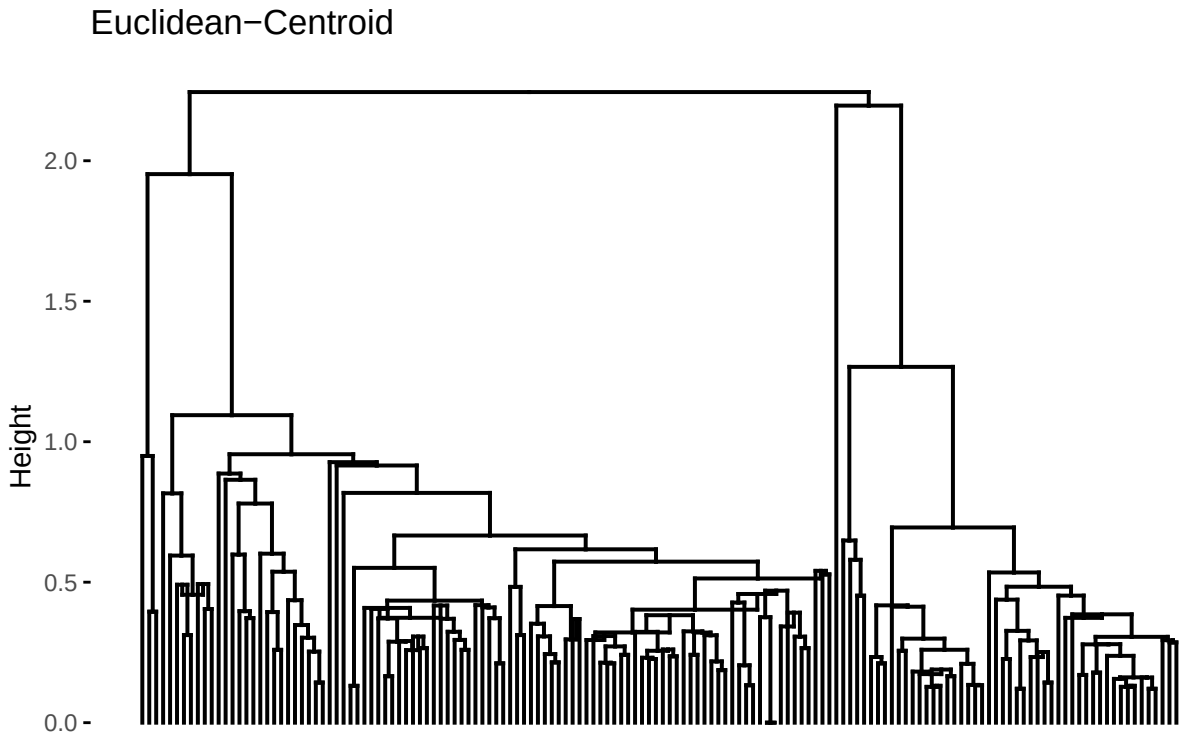

```
fviz_dend(hc_complete, as.ggplot = TRUE, show_labels = FALSE, main='Euclidean-Complete')
```




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




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

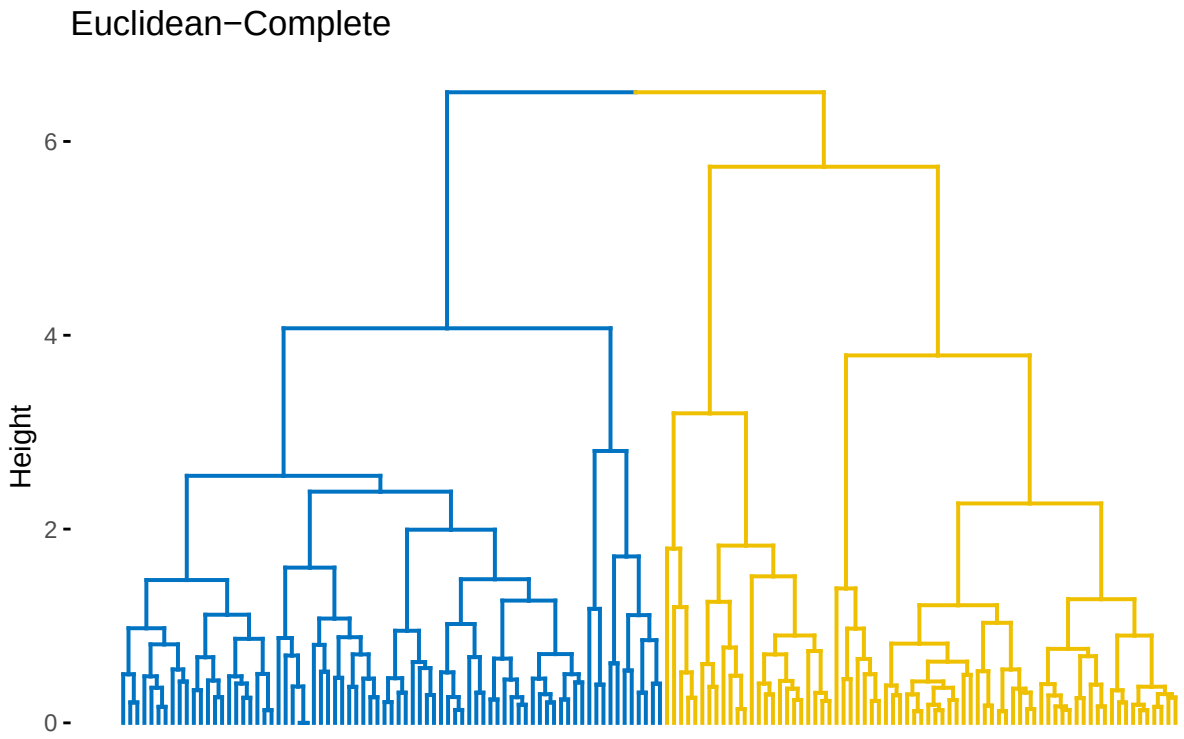


To collect clusters we have to cut the dendrogram using the **cutree** command.

The cut can be performed according to:

- **k:** an integer scalar or vector with the desired number of groups

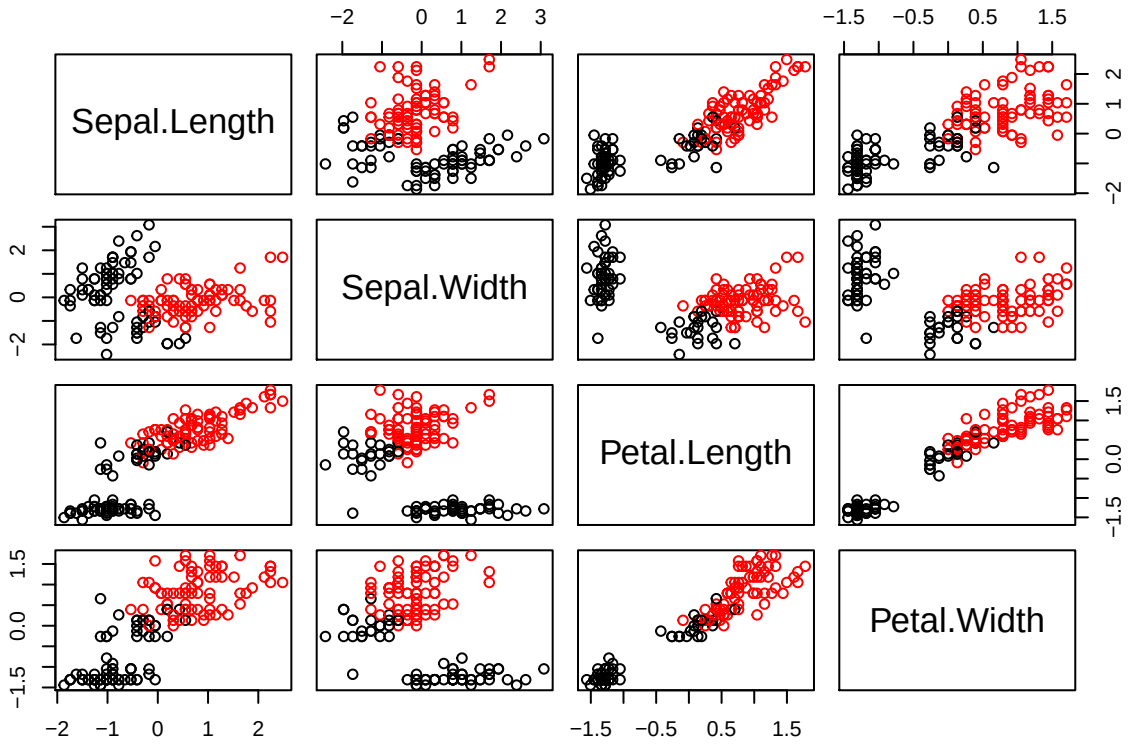
```
cluster_k <- cutree(hc_complete, k = 2) #identifying 2 groups
fviz_dend(hc_complete, k = 2, k_colors = "jco", as.ggplot = TRUE, show_labels = FALSE, main='Euclidean-C
```




```
cluster_k
```

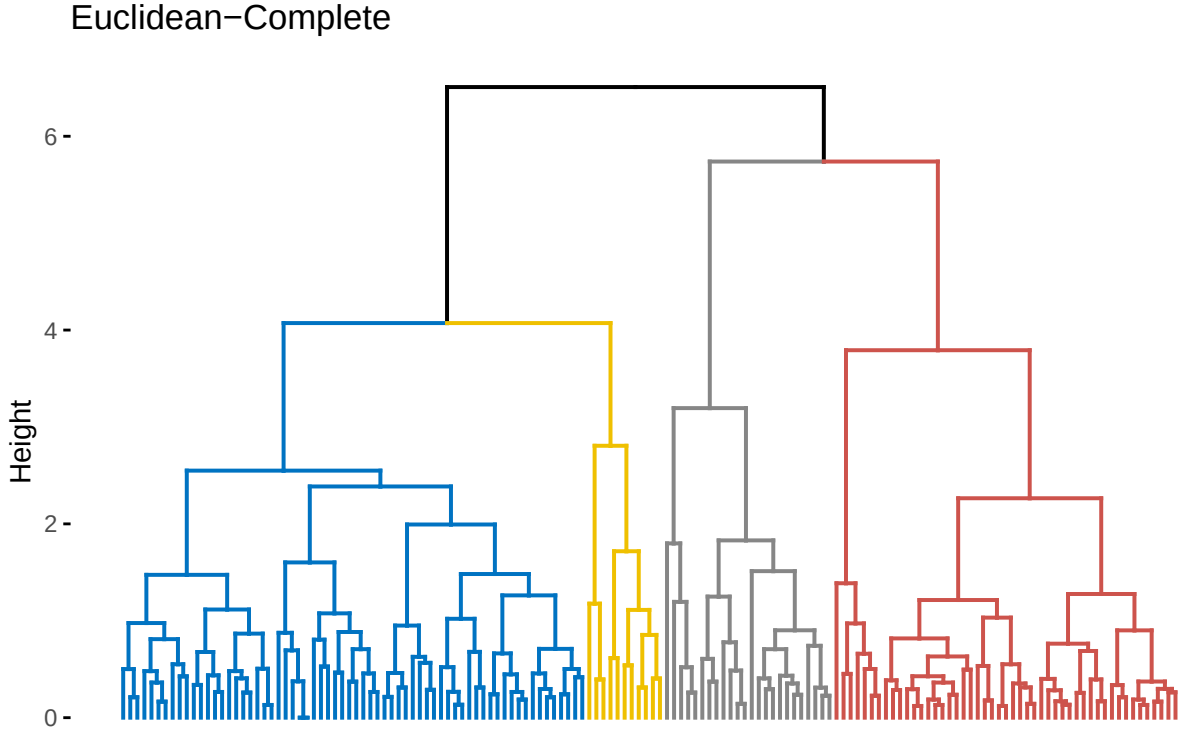
```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 1 2 1 2 1 1 2 1 2 2 2 2 1 1 1 2 2 2 2
##  [75] 2 2 2 2 2 1 1 1 1 2 2 2 2 1 2 1 1 2 1 1 1 2 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2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

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



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

```
cluster_h <- cutree(hc_complete, h = 3.8) #identifying groups below height 3.8
fviz_dend(hc_complete, h = 3.8, k_colors = "jco", as.ggplot = TRUE, show_labels = FALSE, main='Euclidean
```




```
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 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 4 2 4 3 4 3
## [112] 3 3 3 3 3 3 4 4 2 3 3 4 3 3 4 3 3 3 4 4 4 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

Divisive Hierarchical Clustering

Proceeding in an divisive fashion ("top-down"), it generates a sequence of nested partitions of the data – progressively more fine:

- Start from one cluster containing all data point
- At each iteration:
  - Find the largest cluster.
  - Split it.
  - Update the cluster list.
- Iterate until all data points belong to a separate cluster.

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

```
help(diana)
```

We can see that the function requires:

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

Using the previously computed \textbf{eu_iris}.

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

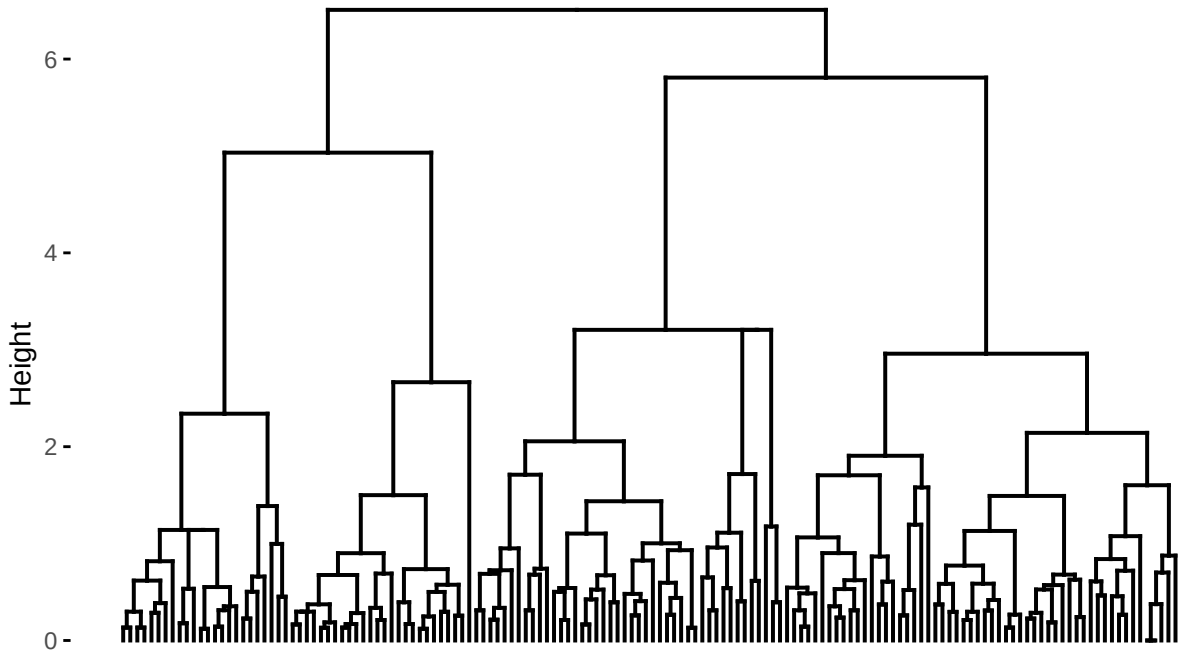
```
## List of 6
##  $ order : int [1:150] 1 28 18 41 21 32 37 44 5 38 ...
##  $ height: num [1:149] 0.133 0.297 0.133 0.618 0.286 ...
##  $ dc    : num 0.94
##  $ merge : int [1:149, 1:2] -102 -8 -11 -10 -129 -18 -128 -3 -1 -81 ...
##  $ diss  : NULL
##  $ call  : language diana(x = eu_iris)
##  - attr(*,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

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

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

To plot and cut the dendrogram we can use

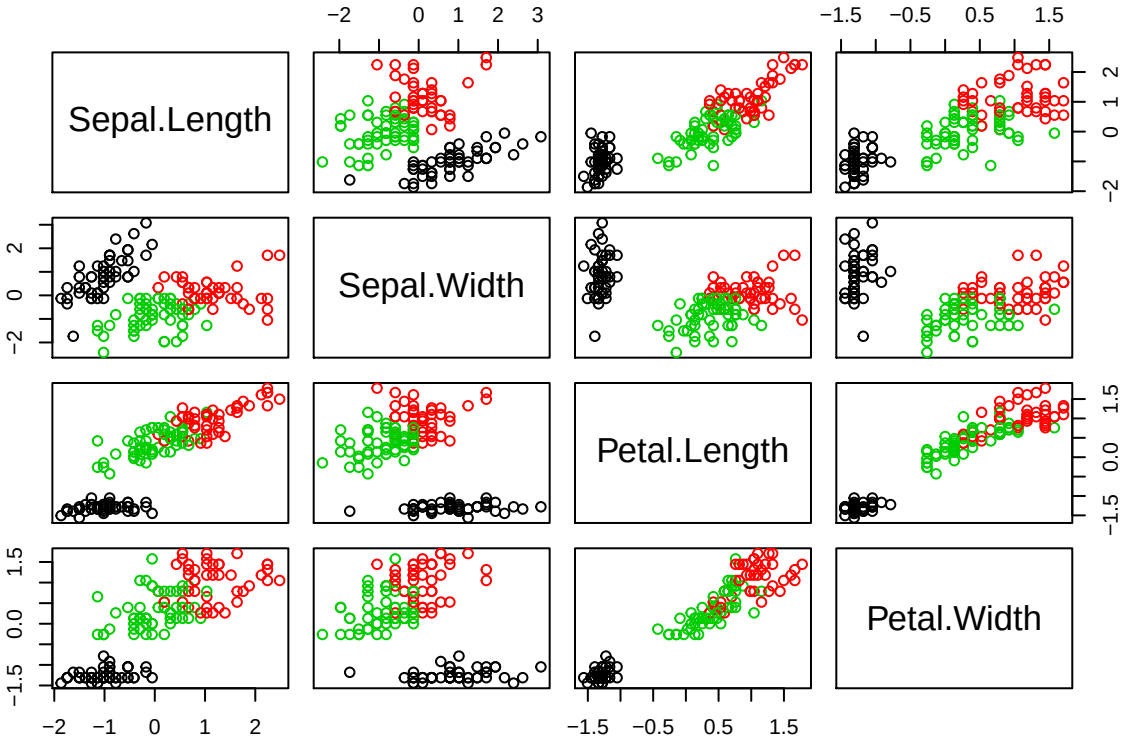
```
fviz_dend(hc_diana, as.ggplot = TRUE, show_labels = FALSE, main='Euclidean-Complete')#plot the dendrogr
```




```
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 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 colored in clusters
```



k-means Clustering

k-means Clustering is a partitioning algorithm that splits the data in k clusters by iteratively computing centroids/moving data points until convergence. To perform the k-means clustering we can use the function **kmeans**.

```
help(kmeans)
```

We can see that the function requires:

- **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.

Using the Iris data set **iris4**:

```
res <- kmeans(iris4, 3)
str(res)
```

```
## List of 9
##  $ cluster     : int [1:150] 2 2 2 2 2 2 2 2 2 2 ...
##  $ centers     : num [1:3, 1:4] -0.0501 -1.0112 1.1322 -0.8804 0.8504 ...
##   ..- 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] 44.1 47.4 47.5
##  $ tot.withinss: num 139
##  $ betweenss   : num 457
##  $ size        : int [1:3] 53 50 47
```

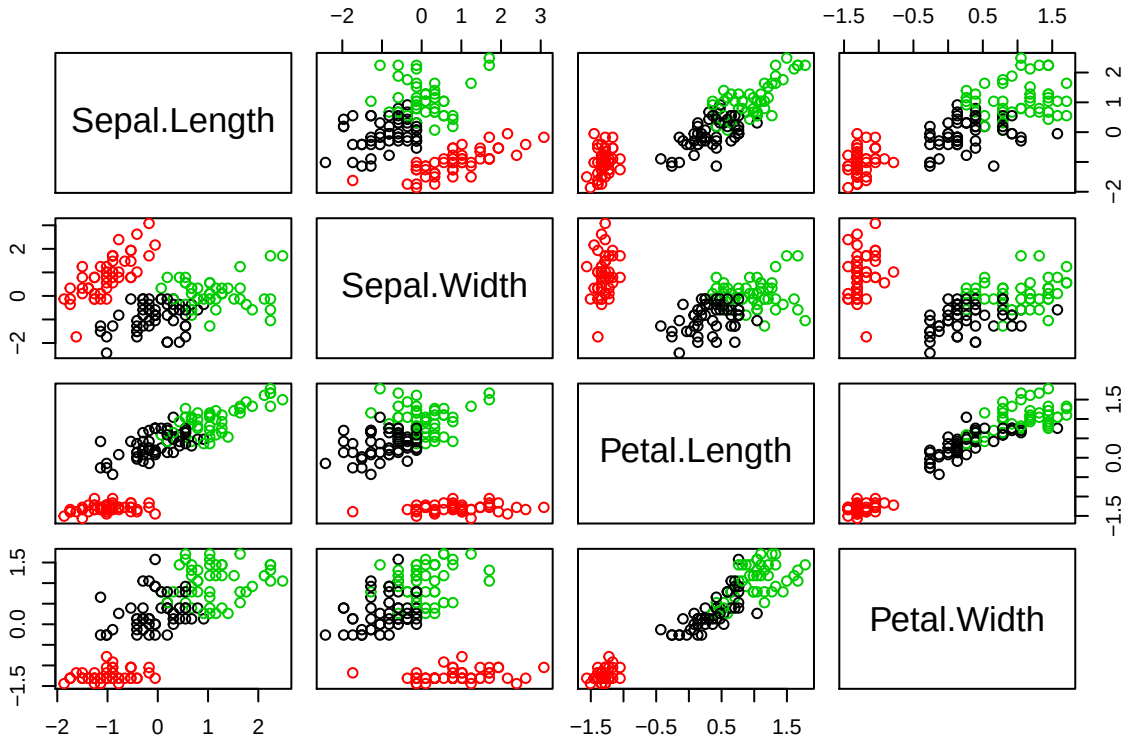
```
##  $ iter        : int 2
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

The clusters are identified in \textbf{res$cluster}:

```
res$cluster
```

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

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



Hierachical Clustering and k-means with one single command

Using the function **eclust** (in **factoextra**) it is possible to perform both the methods. There are also other advantages:

- It can be used to compute hierarchical clustering and partitioning clustering in a single line function call (instead of using two different command)
- Computes automatically the gap statistic for estimating the right number of clusters.
- It provides silhouette information for all partitioning methods and hierarchical clustering
- It draws beautiful and sexy graphs using ggplot2

Let us compute Agglomerative Hierarchical Clustering using eclust.

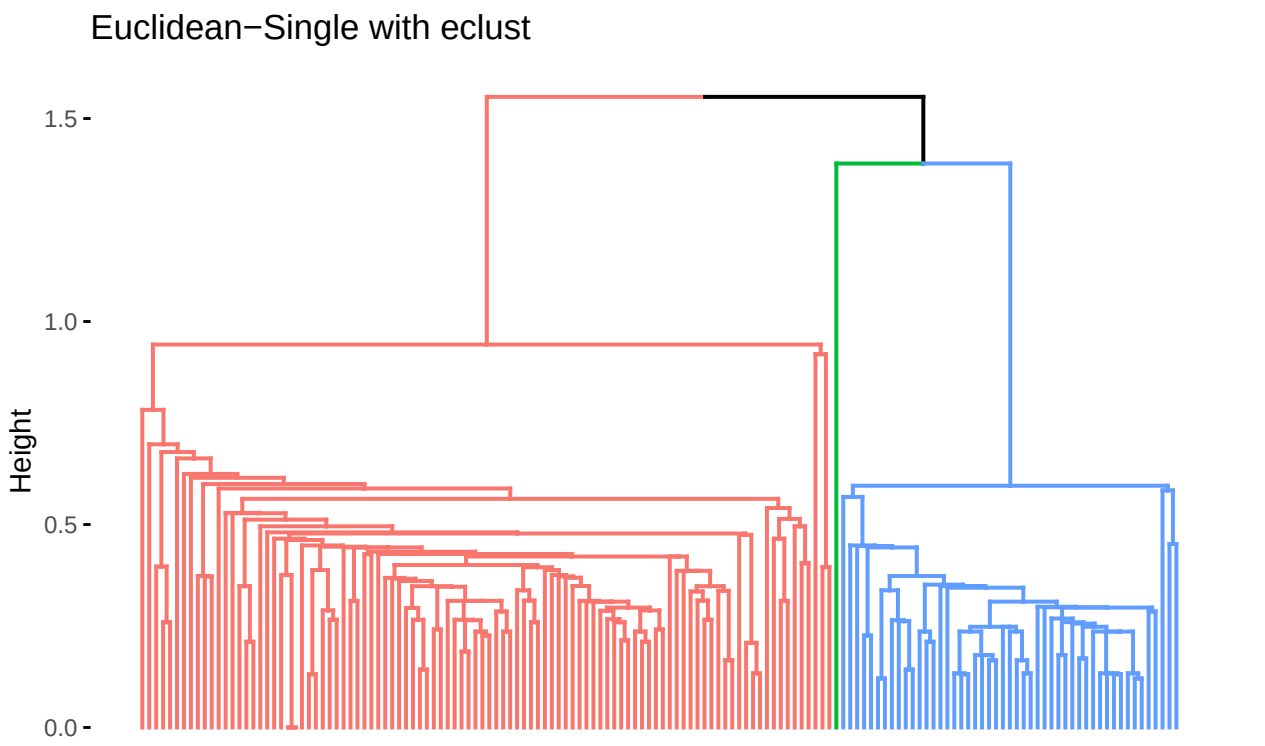
```r
hc_res <- eclust(iris4, "hclust", k = 3, hc_metric = "euclidean", hc_method = "single") # it receives d
str(hc_res)
```

```
## 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"
```

```r
hc_res$cluster
```

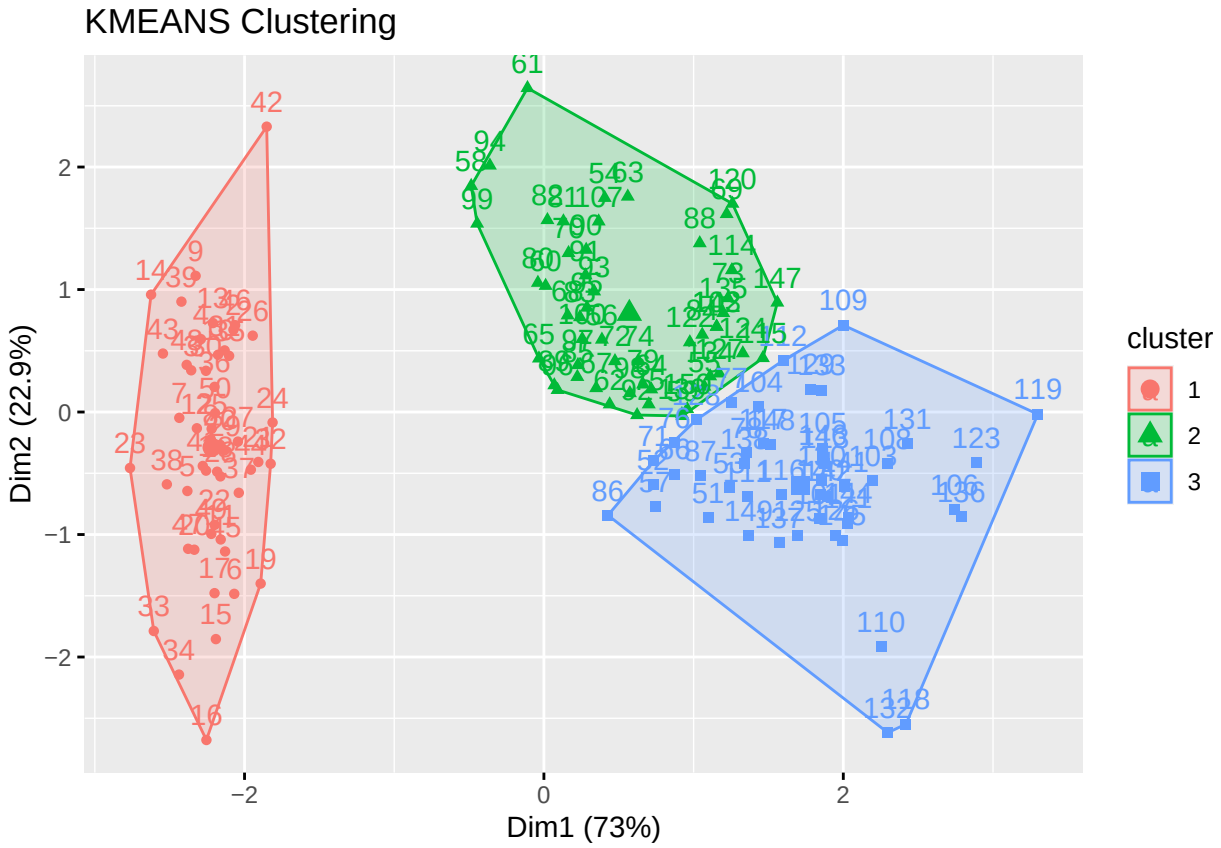
```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 2 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3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

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



Let us compute k-means using eclust. You can notice that the clusters are represented in a 2D scatterplot based on the first two Principal Components (see: 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 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 3 2 3 3 3 3 2 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 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 hight (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:

- **x:** an integer vector with k different integer cluster codes
- **dist:** a dissimilarity object

It returns *"an object, sil, of class silhouette which is an nx3 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."*

Let us use the **silhouette** command with k-means result :

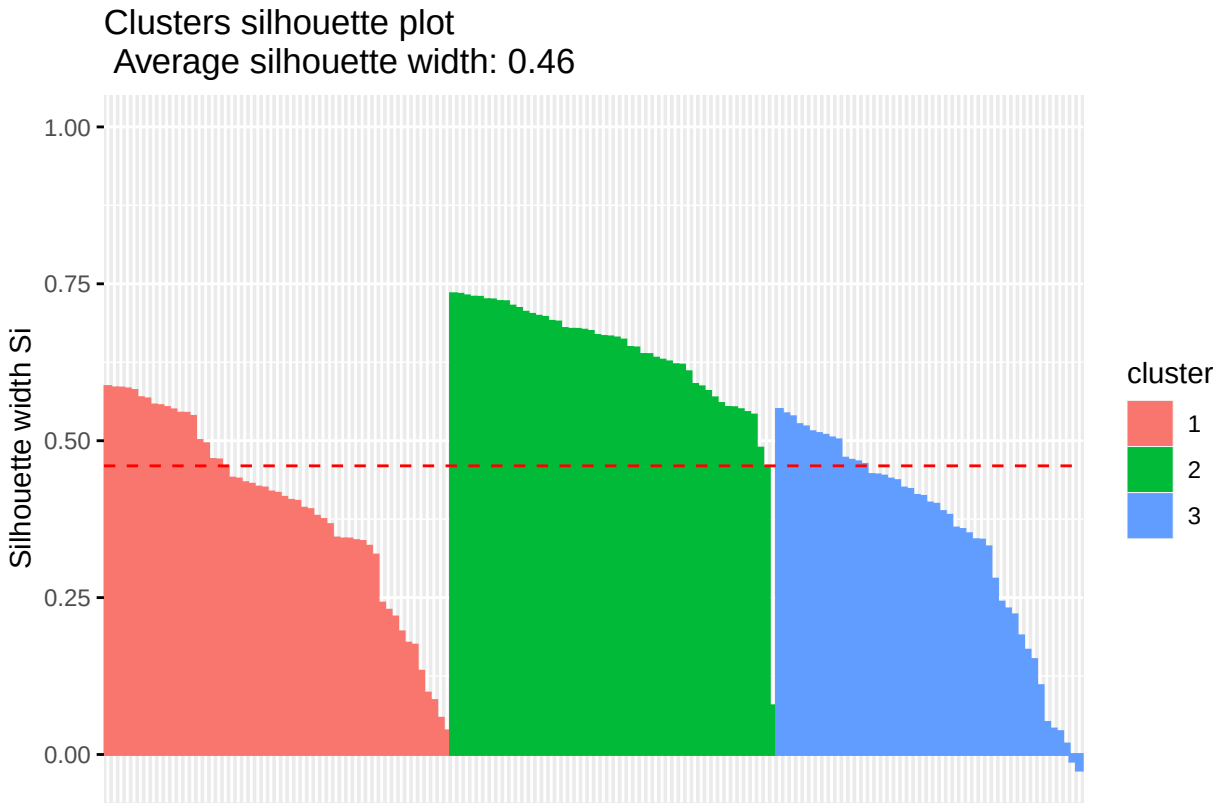
```
distance <- dist(iris4, method="euclidean")
sil <- silhouette(x = res$cluster, dist = distance)
sil[1:5,] # showing the first 5 results
```

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

To get a Silhouette plot we have to work in the factoextra environment.

```
fviz_silhouette(sil)
```

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



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

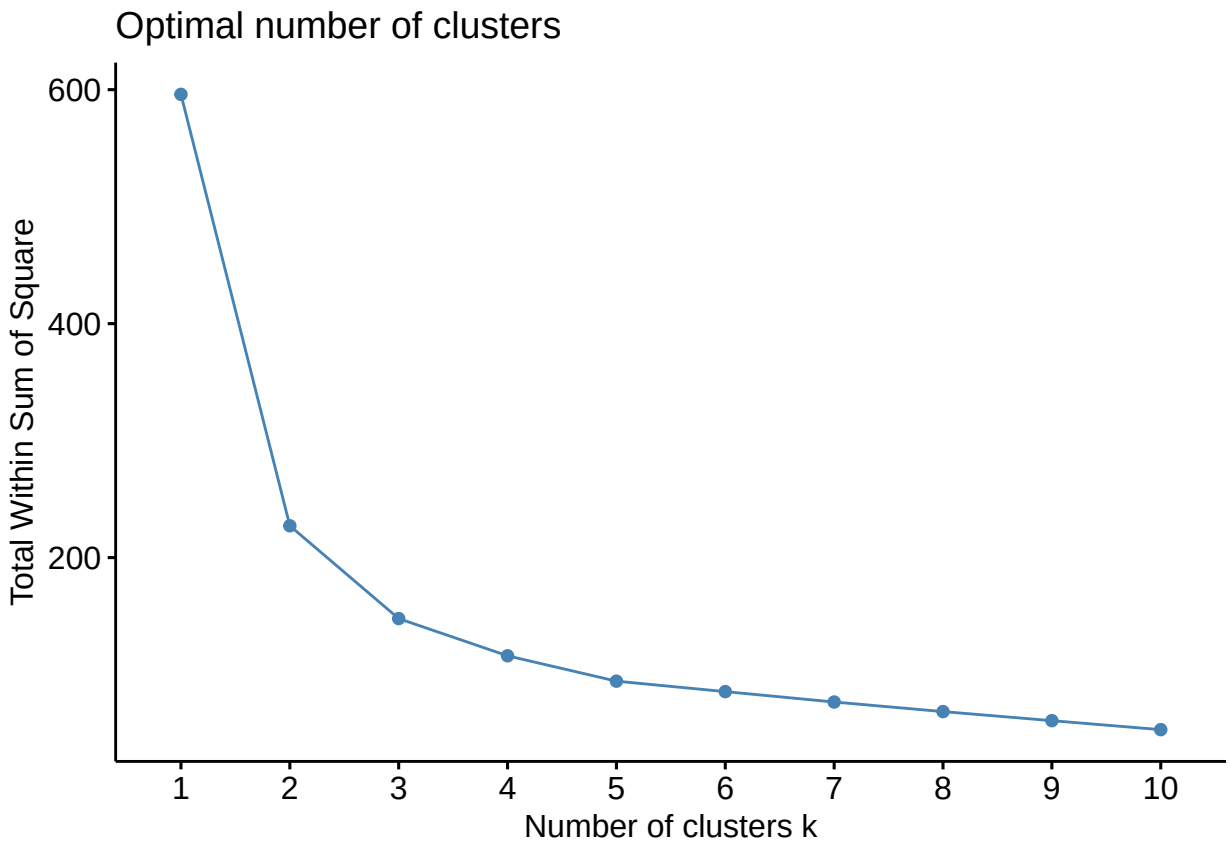
Within cluster dissimilarity/distance

- **Hierarchical:** Dissimilarity levels (heights) at which clusters are formed by cuong.
- ***k*-means:** Within clusters sum of squares (what the algorithm finds a local minimum for)

We can use **fviz-nbclust**.

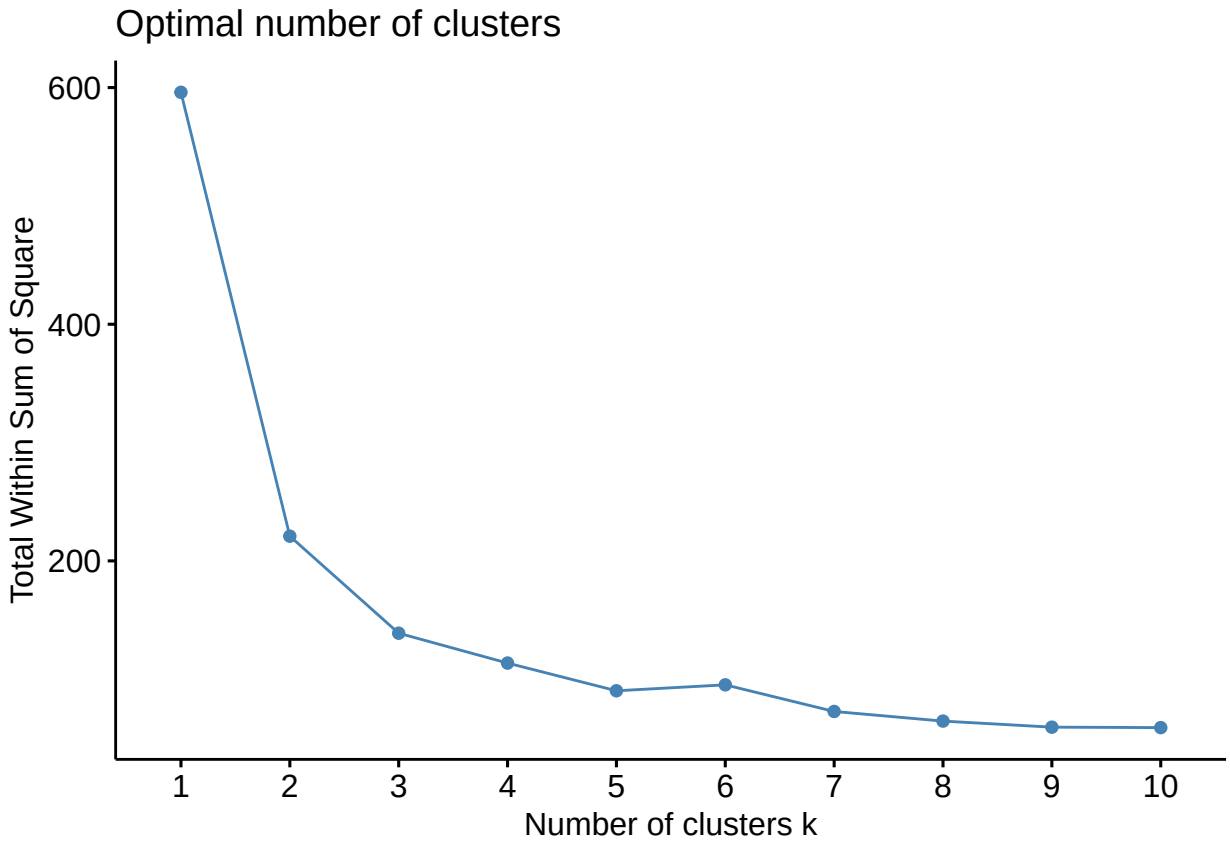
For the Agglomerative Hierarchical Clustering

```
set.seed(123) # same centroids
fviz_nbclust(iris4, hcut, method = "wss")
```



For the *k*-means

```
set.seed(123) # same centroids
fviz_nbclust(iris4, kmeans, method = "wss")
```



set.seed(123) # imposta la selezione dei centroidi casuale uguale per tutti fviz_nbclust(iris4, kmeans, method = “wss”)

Hartigan Index

We can use **NbClust** from **NbClust** library.

For Hierarchical:

```
library(NbClust)
NbClust(iris4,  distance = "euclidean", method = "complete", index='hartigan')
```

```
## $All.index
##        2        3        4        5        6        7        8        9
## 137.0327  33.2185  24.9638  10.6281  12.7058  39.5776  19.9304  32.3094
##       10       11       12       13       14       15
##  16.6788  11.6225   6.9600   4.1462   8.9564   4.8518
##
## $Best.nc
## Number_clusters     Value_Index
##          3.0000        103.8142
##
## $Best.partition
##   [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 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
```

```
## [149] 3 3
```

For k-means:

```
library(NbClust)
NbClust(iris4,  distance = "euclidean", method = "kmeans", index='hartigan')
```

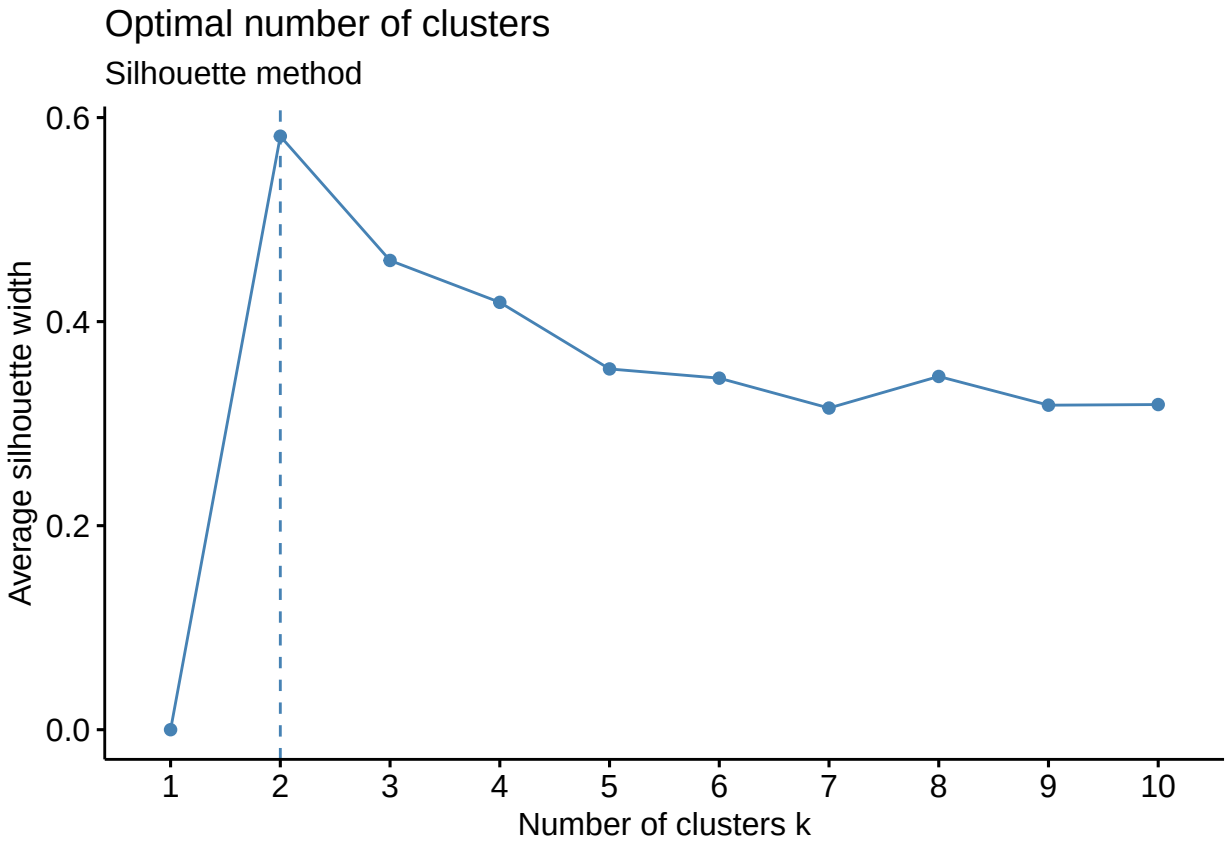
```
## $All.index
##       2       3       4       5       6       7       8       9      10      11
## 87.3699 33.1486 37.4374 19.5911 19.0351 16.2779 20.5630 13.2186  2.3371 14.6665
##      12      13      14      15
##  3.2103  6.7708 31.6603 10.5759
##
## $Best.nc
## Number_clusters     Value_Index
##          3.0000         54.2213
##
## $Best.partition
##   [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
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1
##  [75] 1 2 2 2 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
## [112] 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 2 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2
## [149] 2 1
```

Let us remark that only the data we want to cluster are needed.

Average Silhouette

We can use **fviz-nbclust**.

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



Clustering Additions

Model-based clustering based on parameterized finite Gaussian mixture models can be performed using the **Mclust** library Using the **Mclust** command, models are estimated by EM algorithm initialized by hierarchical model-based agglomerative clustering. The optimal model is then selected according to BIC.

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

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