SLLD - Module 1

Clustering

R.Porcedda, F.Chiaromonte

Sant'Anna School of Advanced Study - Pisa

5/2/2026

Libraries

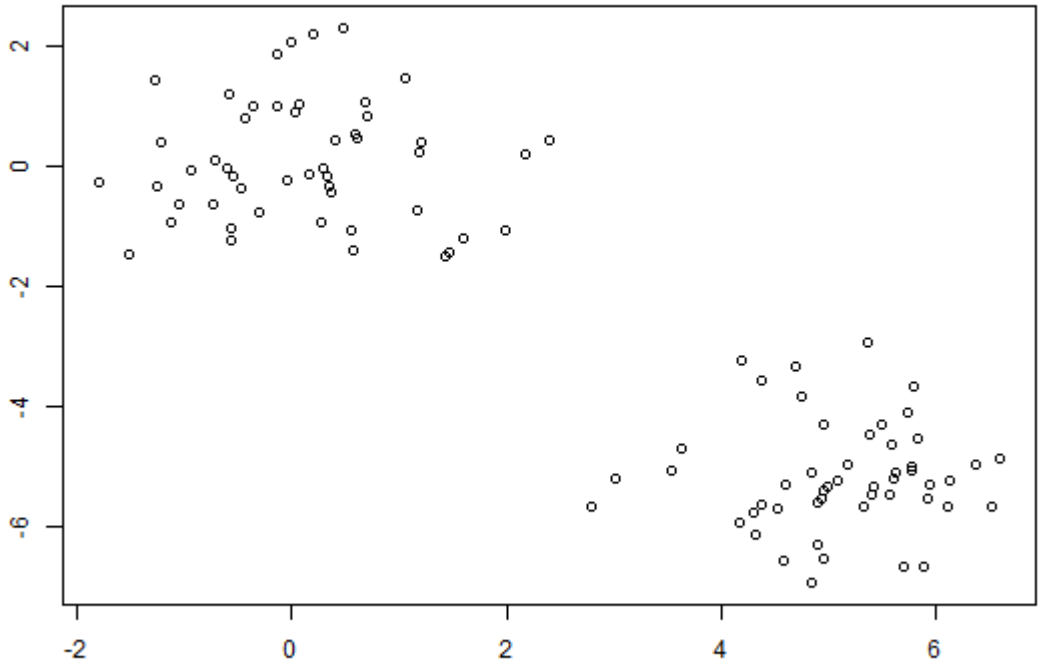
We are going to use

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

We simulate data with a mean shift to obtain two clusters

```
set.seed(1)
x <- matrix(rnorm(100 * 2), ncol = 2)
x[1:50, 1] <- x[1:50, 1] + 5
x[1:50, 2] <- x[1:50, 2] - 5
```

```
plot(x, xlab = "", ylab = "")
```



Hierarchical Clustering

There are two strategies for hierarchical clustering: Agglomerative and Divisive. Here we will focus on the former, which is a “bottom-up” approach which generates a sequence of nested partitions of the data -- progressively less granular:

To perform the Agglomerative Hierarchical Clustering we can use the function **hclust()** in *R*. It requires

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

First, we use the function **dist()** to create a dissimilarity matrix based on the **Euclidean distance** in the following way

```
eu_d <- dist(x, method='euclidean')
as.matrix(eu_d)[1:5,1:5] # distances between the first 5 obs.
```

```
##           1         2         3        4         5
## 1 0.0000000 1.0464896 0.3580172 2.354146 0.9565738
## 2 1.0464896 0.0000000 1.3954196 1.416388 0.7118062
## 3 0.3580172 1.3954196 0.0000000 2.655556 1.1930010
## 4 2.3541462 1.4163884 2.6555556 0.000000 1.5041682
## 5 0.9565738 0.7118062 1.1930010 1.504168 0.0000000
```

Now we are ready to create the hierarchical structure, based on the **single linkage** method.

```
hc_single <- hclust(eu_d, method='single') # change 'single' with
# 'complete', 'average' or 'centroid' to perform other linkages

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

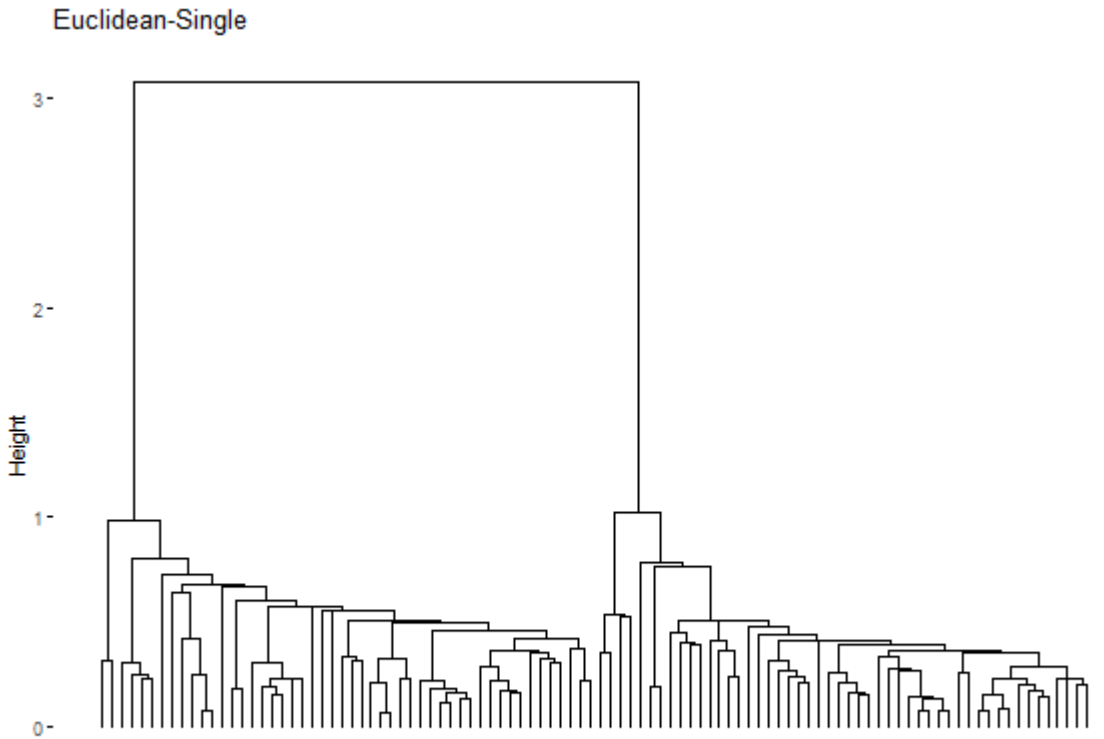
```
## List of 7
##  $ merge      : int [1:99, 1:2] -73 -55 -40 -32 -16 -20 -53 -76 4 -23 ...
##  $ height     : num [1:99] 0.0662 0.0709 0.0745 0.0746 0.0783 ...
##  $ order      : int [1:100] 61 70 71 60 66 78 97 93 56 95 ...
##  $ labels     : NULL
##  $ method     : chr "single"
##  $ call       : language hclust(d = eu_d, method = "single")
##  $ dist.method: chr "euclidean"
##  - attr(*, "class")= chr "hclust"
```

```
head(hc_single$merge, 5) # see the first 5 aggregations
```

```
##      [,1] [,2]
## [1,]  -73  -85
## [2,]  -55  -68
## [3,]  -40  -48
## [4,]  -32  -38
## [5,]  -16  -17
```

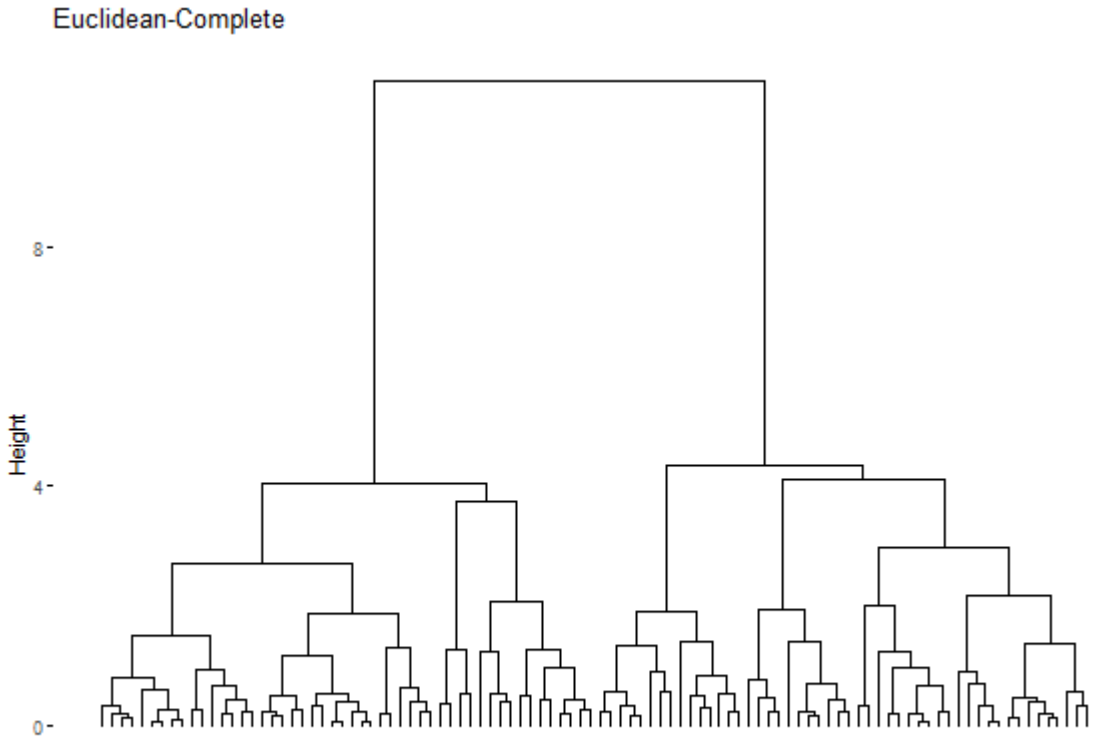
The hierarchy is represented through a dendrogram: a plot illustrating a sequence of data partitions into clusters.

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



Here the **complete linkage** method

```
hc_complete <- hclust(eu_d, method='complete')
fviz_dend(hc_complete,   as.ggplot = TRUE,
          show_labels = FALSE, main='Euclidean-Complete')
```



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

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

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

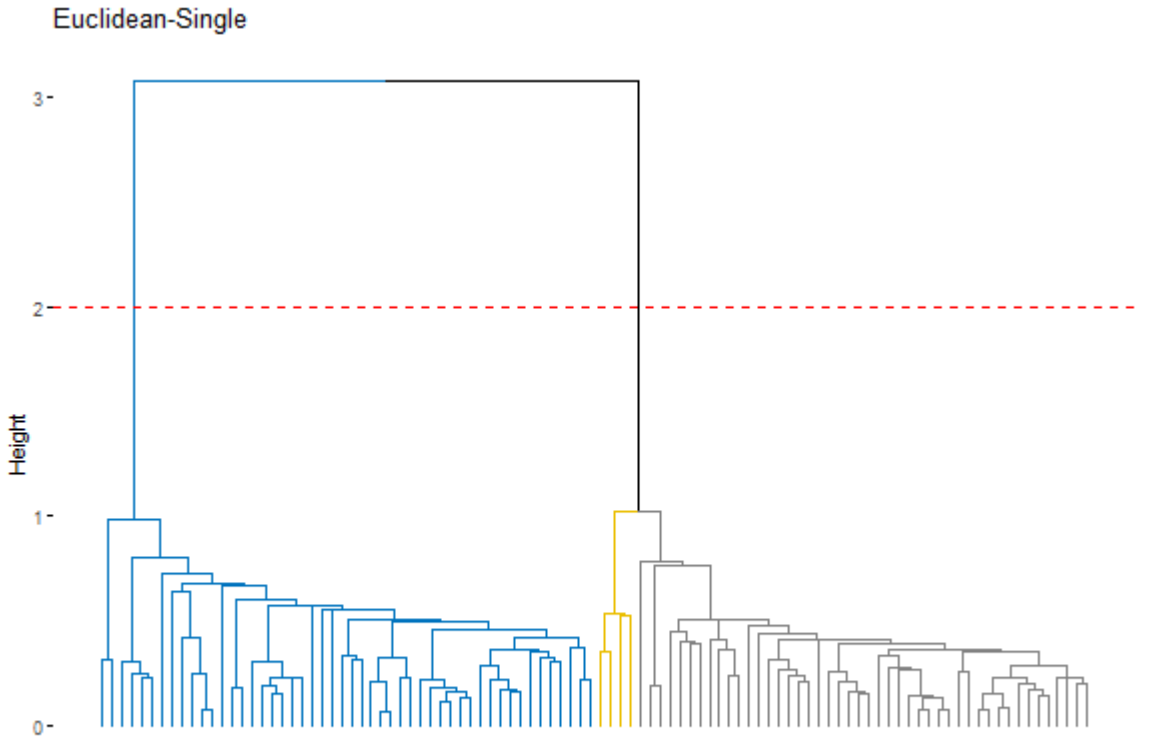
```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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

```
clHeight <- 2
cluster_h <- cutree(hc_single, 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 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 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
```

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



K-Means Clustering

The function **kmeans()** performs K-means clustering in *R*. It requires:

- **x:** numeric matrix of data
- **centers:** either the number of clusters, say k, or a set of initial (distinct) cluster centers
- **nstart:** initial cluster assignments

Note 1: 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!!

We now perform K-means clustering with $K = 2$ and $nstart = 5$

Note 2: Here the role of **nstart** is less impactful given the way in which the data was generated!

```
set.seed(1)
km.2 <- kmeans(x, 2, nstart=5)
```

```
km.2$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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 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
```

```
km.2$centers
```

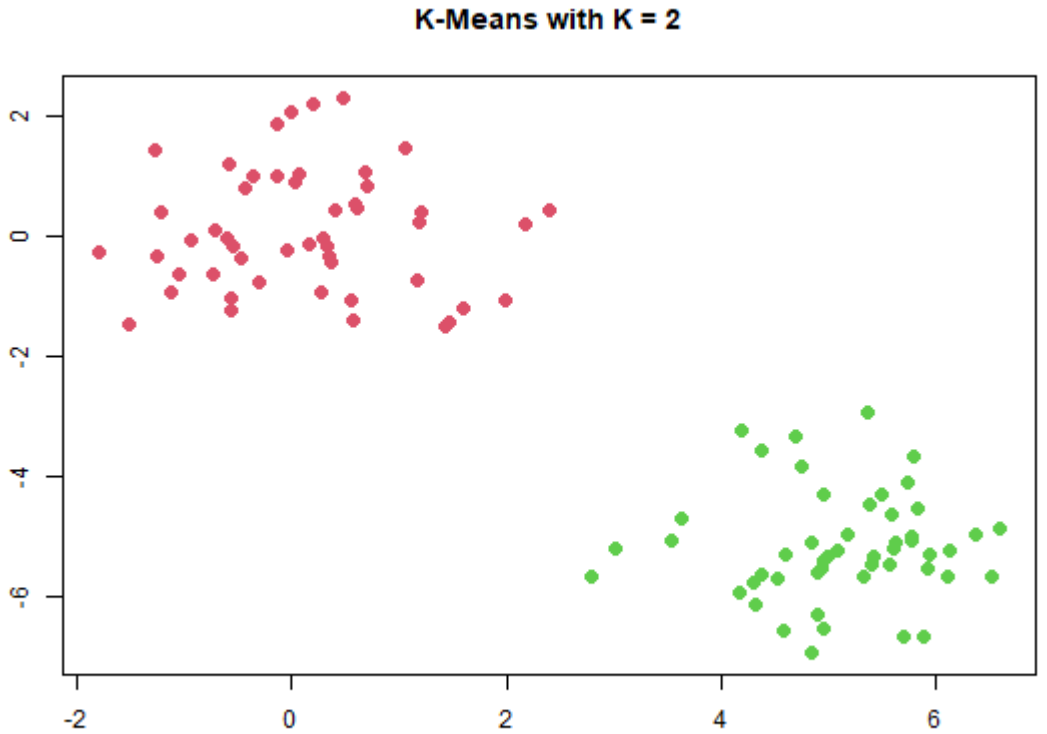
```
##        [,1]        [,2]
## 1 0.1173265  0.07686929
## 2 5.1004483 -5.15248544
```

```
km.2$withinss
```

```
## [1] 95.84634 73.53660
```

cluster:A vector of integers (from 1:k) indicating the cluster to which each point is allocated. **centers:**A matrix of cluster centres. **totss:**The total sum of squares. **withinss:**Vector of within-cluster sum of squares, one component per cluster. **tot.withinss:**Total within-cluster sum of squares, i.e. sum(withinss). **betweenss:**The between-cluster sum of squares, i.e. totss-tot.withinss. **size:**The number of points in each cluster. **iter:**The number of (outer) iterations. **ifault:**integer: indicator of a possible algorithm problem – for experts.

```
plot(x, col = (km.2$cluster + 1),
main = "K-Means with K = 2",
xlab = "", ylab = "", pch = 20, cex =2)
```



Here we know that there are 2 clusters! However, for real data we do not known the true number of clusters. What happens if we try $K = 3$? In general, how can we evaluate clustering?

Now we perform K-means with $K = 3$

```
set.seed(1)
km.3 <- kmeans(x, 3, nstart=20)
km.3$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 3
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 2 1 1 1 1 1 2 1 1 2 2 1 2 2 1 2 1 1 1 2 2 1 2 1
##  [75] 1 1 2 2 2 2 1 2 2 1 2 1 2 1 1 1 1 2 1 2 1 1 2 1 1 1
```

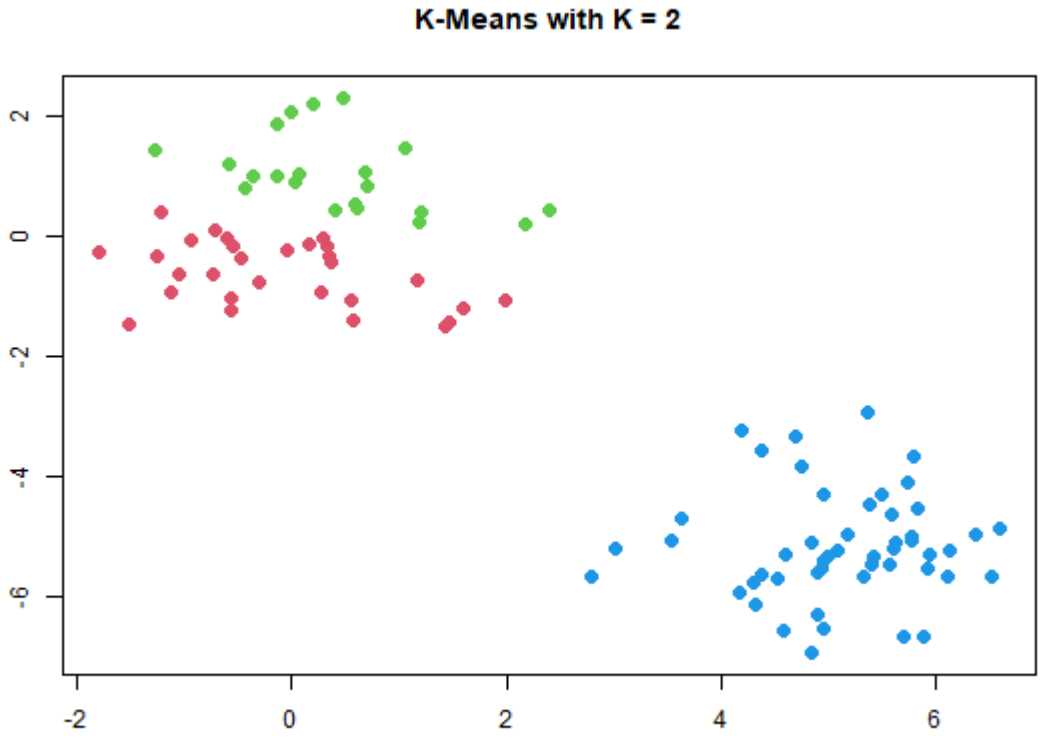
```
km.3$centers
```

```
##         [,1]       [,2]
## 1 -0.1024339 -0.6202009
## 2  0.4208051  1.0394900
## 3  5.1004483 -5.1524854
```

```
km.3$withinss
```

```
## [1] 35.41052 23.55048 73.53660
```

```
plot(x, col = (km.3$cluster + 1),
main = "K-Means with K = 2",
xlab = "", ylab = "", pch = 20, cex = 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. It requires:

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

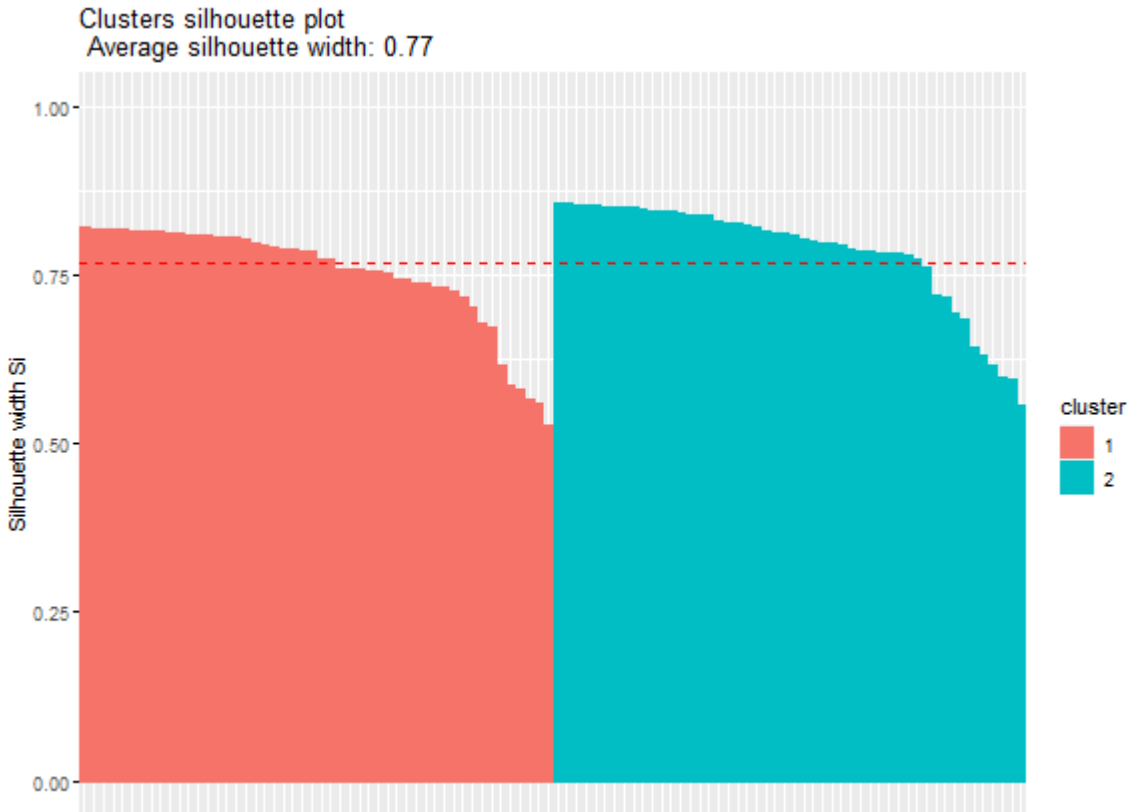
```
sil.2 <- silhouette(x = km.2$cluster, dist = eu_d)
sil.2[1:5,] # showing the first 5 results
```

```
##      cluster neighbor sil_width
## [1,]       2        1 0.8099942
## [2,]       2        1 0.8458737
## [3,]       2        1 0.7835822
## [4,]       2        1 0.7805404
## [5,]       2        1 0.8498846
```

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

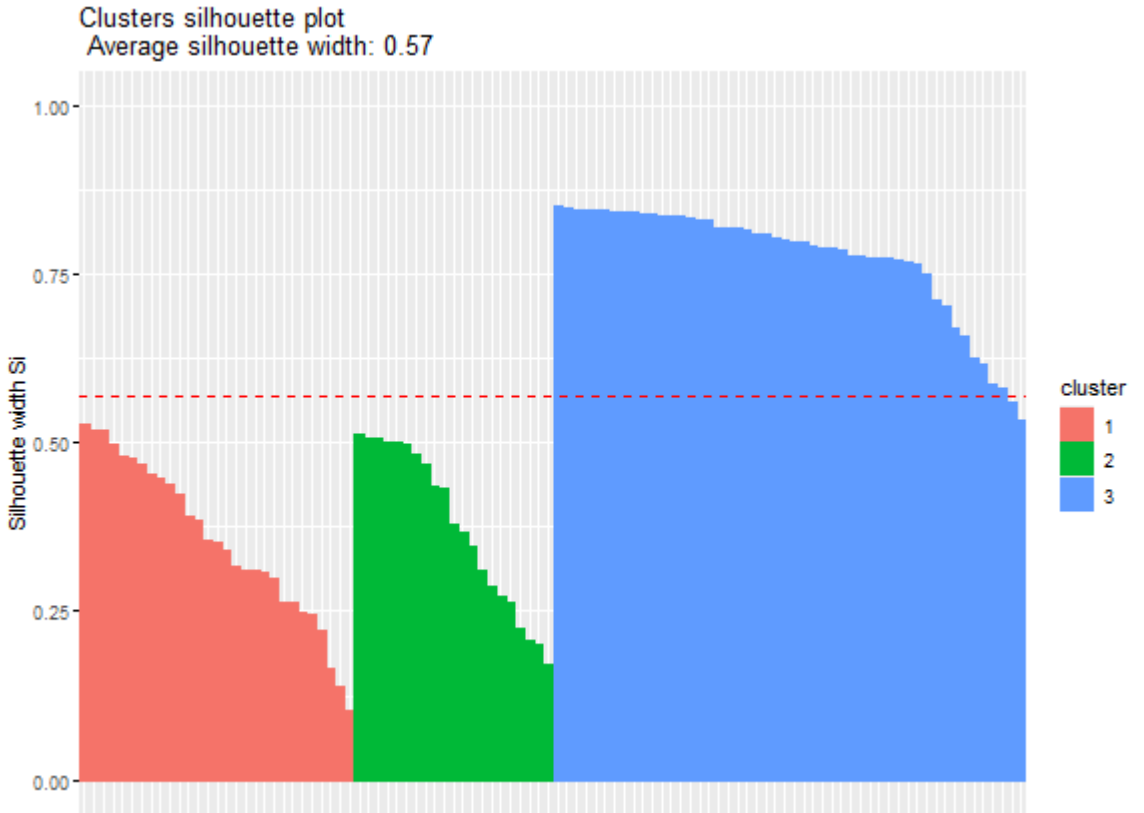
```
fviz_silhouette(sil.2)
```

```
##   cluster size ave.sil.width
## 1       1   50          0.75
## 2       2   50          0.78
```



```
sil.3 <- silhouette(x = km.3$cluster, dist = eu_d)
fviz_silhouette(sil.3)
```

```
##   cluster size ave.sil.width
## 1       1   29          0.35
## 2       2   21          0.37
## 3       3   50          0.77
```



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

Remember, we can do these analyzes for both

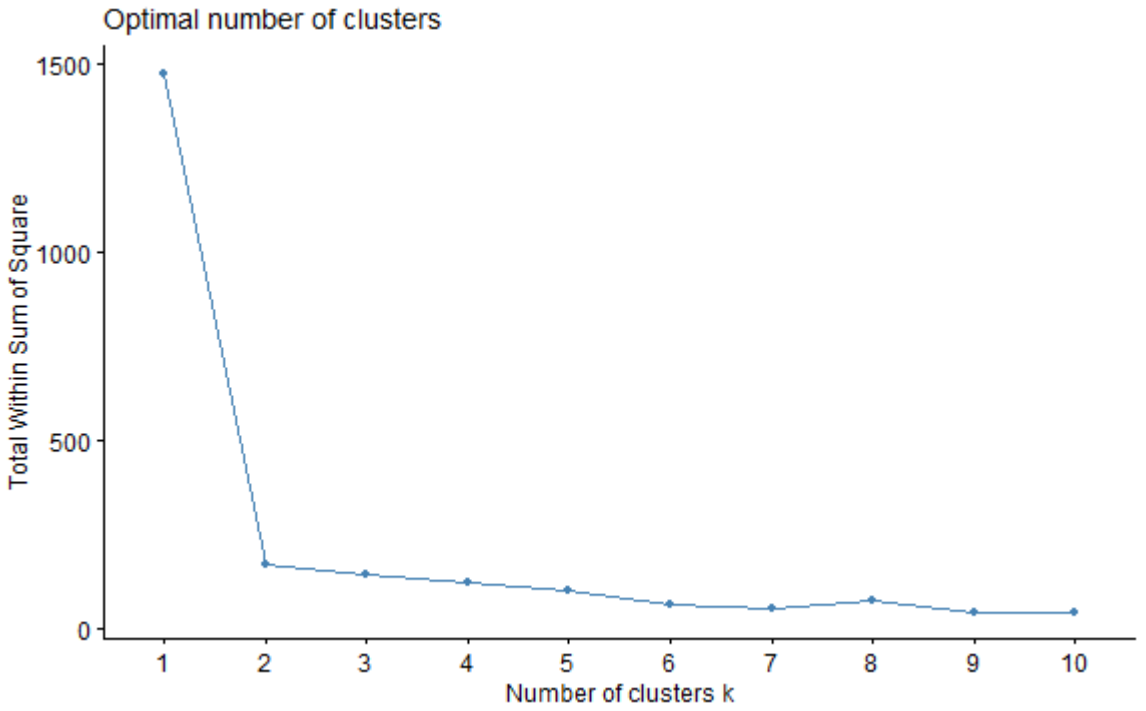
- Hierarchical: Dissimilarity levels (heights) at which clusters are formed
- k-means: Within clusters sum of squares (it is guaranteed to be a local minimum for any given random initialization)

Here we will focus on k-means!

Within cluster dissimilarity/distance

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

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

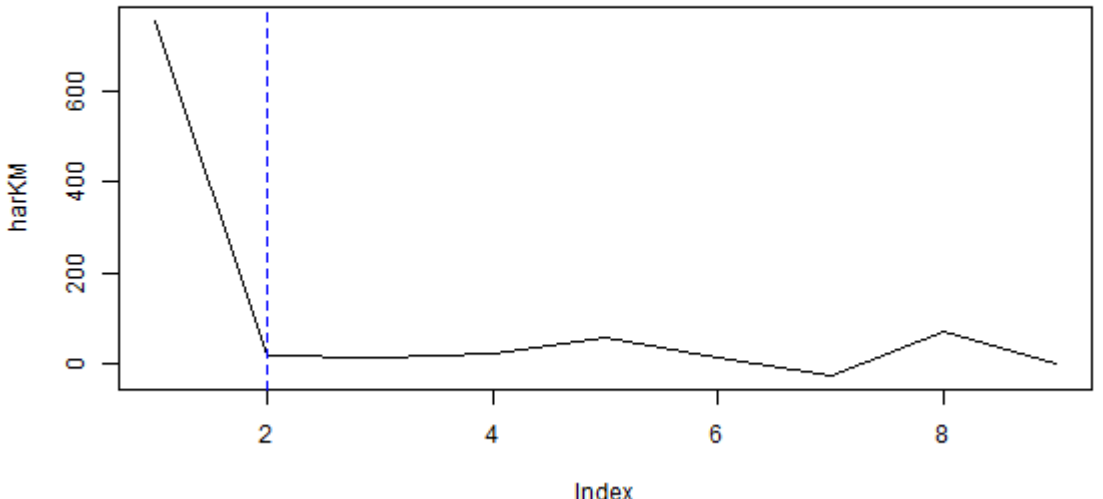



```
# replace 'kmeans' with 'hcut' for the Agglomerative Hierarchical
```

Hartigan Index

We can use the within cluster dissimilarity/distance previously calculated to obtain the Hartigan index...homemade function:)

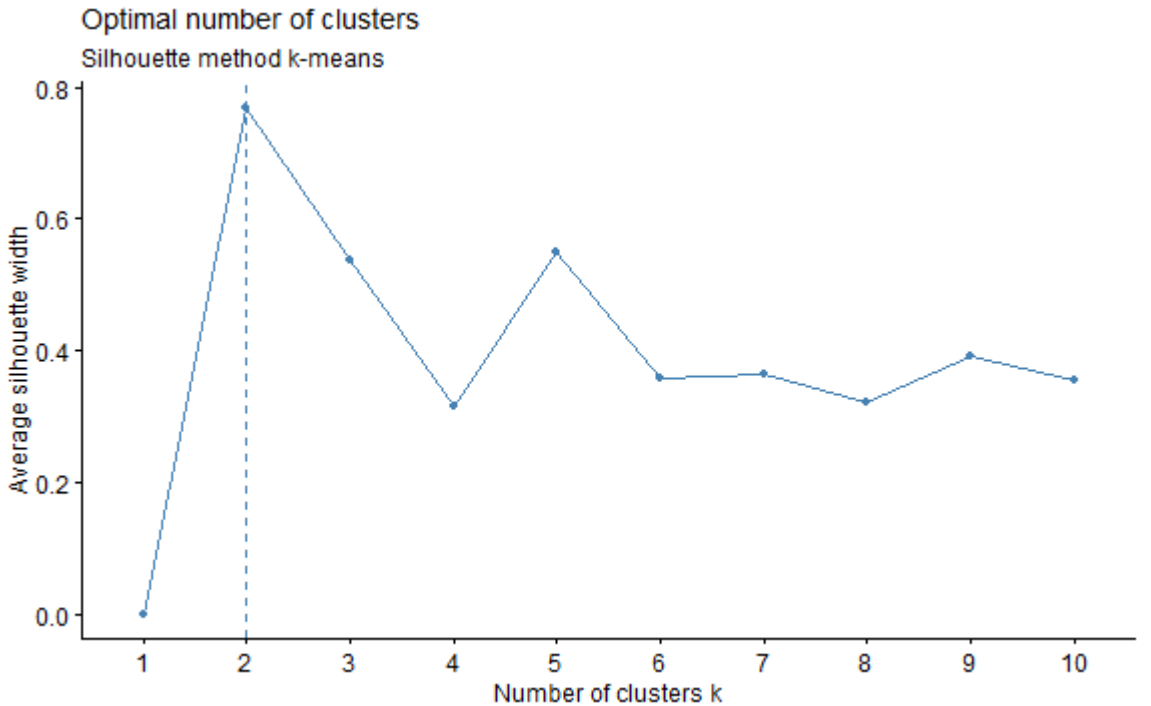
```
WSS_km<-fviz_nbclust(x, kmeans, method = "wss")$data[,2]
harKM <- NULL
for (i in 1:(length(WSS_km)-1)) {
  harKM[i] <- (nrow(x)-i-1)*(WSS_km[i]-WSS_km[i+1])/WSS_km[i+1]
  Best.nc_km<-which.max(harKM)+1
}
plot(harKM,type="l")
abline(v=Best.nc_km, col="blue", lty=2)
```



Average Silhouette

We can use **fviz-nbclust**.

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




```
# replace 'kmeans' with 'hcut' for the Agglomerative Hierarchical
```

Now it's your turn!!!