Clustering methods

Hierarchical, Partitioning, and Model-based Clustering

Denaldo Lapi, Francesco Aristei, Samy Chouiti

05 June 2022

Outline

Delete all the possible objects of R that could have been left in memory:

```
rm(list = ls())
```

At first, let's load the libraries we'll need:

- *dplyr* and *ggplot2* will help us with data manipulation and graphs, respectively.
- *kableExtra* will help us to print beautiful tables.
- *gridExtra* for arranging the layout of the graphs.
- *stats* computes hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it (*hclust*).
- *cluster* to apply the *agnes* (agglomerative hierarchical clustering) and the *diana* (divisive hierarchical clustering).
- *gplots* computes heatmaps for visualizing the distance matrix.
- *factoextra* for MVA methods and graph the clustering structures.
- *FactoMineR* computes hierarchical clustering on principal components.

```
library(dplyr)
library(ggplot2)
library(kableExtra)
library(gridExtra)
library(stats)
library(gplots)
library(cluster)
library(factoextra)
library(FactoMineR)
```

Table 1: Prostate cancer data set (sample of 20)

age	wt	pf	hx	sbp	dbp	ekg	hg	sz	sg	ap	bm
79	79	2	0	15	7	1	13.29883	16	13	28.8984375	1
65	99	0	0	13	8	0	14.39844	10	9	0.5999756	0
75	88	0	1	14	6	0	13.39844	31	13	21.5976562	1
70	94	0	0	15	7	1	13.69922	24	7	0.7999268	0
76	103	0	1	13	7	5	12.50000	4	8	0.5000000	0
70	91	0	1	16	9	3	15.89844	7	9	1.0000000	0
54	89	0	0	15	6	2	14.79883	8	13	1.1999512	1
74	113	0	1	11	8	4	14.19922	41	13	2.2998047	0
73	106	0	1	16	9	0	14.69922	11	11	36.8984375	0
74	107	0	0	14	9	5	14.39844	6	9	0.2999878	0

1 Exploratory data analysis

The dataset we are going to analyze is about prostate cancer. It consists of p=12 variables of mixed type (8 continuous and 4 categorical) measured on a group of n=50 prostate cancer patients. These patients have either stage 3 or stage 4 prostate cancer.

Let's load the data from the R image 'Prostate.RData'.

```
load("Prostate.RData")
```

Check dimension:

```
dim(Prostate)
```

```
## [1] 50 12
```

We have 50 rows and 12 columns

Check the structure:

```
str(Prostate)
```

```
## tibble [50 x 12] (S3: tbl_df/tbl/data.frame)
##  $ age: num [1:50] 80 80 66 71 72 79 73 76 72 78 ...
##  $ wt : num [1:50] 105 95 116 98 101 79 80 99 88 107 ...
##  $ pf : Factor w/ 4 levels "0","1","2","3": 3 1 1 1 1 3 1 1 1 1 ...
##  $ hx : Factor w/ 2 levels "0","1": 2 2 1 1 1 1 2 2 1 1 ...
##  $ sbp: num [1:50] 14 16 12 19 14 15 13 15 14 14 ...
##  $ dbp: num [1:50] 7 10 8 10 10 7 8 9 7 9 ...
##  $ ekg: Factor w/ 7 levels "0","1","2","3",..: 6 5 1 1 3 2 5 5 5 1 ...
##  $ hg : num [1:50] 8.5 14.5 14.9 15.1 16.4 ...
##  $ sz : num [1:50] 6 7 18 10 1 16 61 9 7 42 ...
##  $ sg : num [1:50] 5 9 12 11 9 13 12 11 12 11 ...
##  $ ap : num [1:50] 160 0.9 58.1 0.6 0.8 ...
##  $ bm : Factor w/ 2 levels "0","1": 2 1 1 1 1 2 1 1 1 1 ...
```

We can print a portion (a sample of 10) of the table using kable and the pipe operator:

```
Prostate %>%
  sample_n(., 10, replace=FALSE) %>%
  kbl(caption = "Prostate cancer data set (sample of 20)") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Let's now visualize some basic statistics on each of the data frame's columns with *summary*:

Table 2: Basic statistics. Prostate cancer data set

	age	wt	pf	hx	sbp	dbp	ekg	hg	sz	sg
	Min. :52.00	Min. : 79.00	0:45	0:29	Min. :10.00	Min. : 6.00	0:19	Min. : 8.50	Min. : 0.0	Min. :
	1st Qu.:71.00	1st Qu.: 93.25	1: 3	1:21	1st Qu.:13.00	1st Qu.: 7.00	1: 6	1st Qu.:12.27	1st Qu.: 7.0	1st Qu
	Median :74.00	Median : 99.00	2: 2	NA	Median :14.00	Median : 8.00	2: 3	Median :13.50	Median :12.5	Media
	Mean :72.76	Mean : 98.44	3: 0	NA	Mean :14.38	Mean : 7.98	3: 3	Mean :13.35	Mean :15.9	Mean
	3rd Qu.:77.00	3rd Qu.:104.75	NA	NA	3rd Qu.:16.00	3rd Qu.: 9.00	4:13	3rd Qu.:14.70	3rd Qu.:20.0	3rd Qu
	Max. :84.00	Max. :119.00	NA	NA	Max. :19.00	Max. :11.00	5: 6	Max. :18.20	Max. :61.0	Max.
	NA	NA	NA	NA	NA	NA	6: 0	NA	NA	NA

```
Prostate %>%
  summary(.) %>%
  kbl(caption = "Basic statistics. Prostate cancer data set") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Since clustering methods we analyzed work by considering a continuous domain, i.e. continuous variables, we'll remove the categorical ones. Another possibility could be to encode them in some numeric way, but we should take care of the distance measure to use in the hierarchical and k-means algorithms, which is difficult to express in case of categorical variables (euclidean distances doesn't make a lot of sense in the case of categorical variables, we should considere other types of distances).

Let's select only the numerical columns:

```
library(purrr)
Prostate_reduced = Prostate[, c(1,2,5,6,8,9,10,11)]
```

Let's now check again the structure:

```
str(Prostate_reduced)
```

```
## tibble [50 x 8] (S3: tbl_df/tbl/data.frame)
##  $ age: num [1:50] 80 80 66 71 72 79 73 76 72 78 ...
##  $ wt : num [1:50] 105 95 116 98 101 79 80 99 88 107 ...
##  $ sbp: num [1:50] 14 16 12 19 14 15 13 15 14 14 ...
##  $ dbp: num [1:50] 7 10 8 10 10 7 8 9 7 9 ...
##  $ hg : num [1:50] 8.5 14.5 14.9 15.1 16.4 ...
##  $ sz : num [1:50] 6 7 18 10 1 16 61 9 7 42 ...
##  $ sg : num [1:50] 5 9 12 11 9 13 12 11 12 11 ...
##  $ ap : num [1:50] 160 0.9 58.1 0.6 0.8 ...
```

And the dimensions:

```
dim(Prostate_reduced)
```

```
## [1] 50  8
```

1.0.1 Check for missing values

Let's check for NA values:

```
colSums((is.na((Prostate_reduced))))
```

```
## age  wt sbp dbp  hg  sz  sg  ap
##   0   0   0   0   0   0   0   0
```

There are no missing values!

1.0.2 Outliers

One way to check for multivariate outliers is to use the Malhanobis' distance . It can be thought of as a metric for estimating how far each observation is from the center of all the variables' distributions (i.e. the centroid in the multivariate space).

We'll use the *chemometrics* package, which contains a function ('Moutlier') for calculating and plotting both the "Mahalanobis"' distance and a robust version of the "Mahalanobis"' distance.

At first, let's calculate the "Mahalanobis"' distances using the 'Moutlier' function, to which we provide as parameters the numeric data frame, the quantile cutoff point beyond which we want to identify points as outliers, and whether or not we want a plot:

```
#install.packages("chemometrics")
library(chemometrics)
md <- Moutlier(Prostate_reduced, quantile = 0.975, plot=FALSE)
```

The function returns the cutoff value for the outliers:

```
md$cutoff
```

```
## [1] 4.187427
```

We simply use the 'which' function to identify which cases are outliers according to the 'cutoff' value and in this way we obtain the outliers' indexes:

```
outliers <- which(md$md > md$cutoff)
head(outliers, 10) # show first 10 outliers according to Malhanobis distance
```

```
## [1]  7 13
```

Another approach for identifying multivariate outliers (non-parametric approach) is to use the LOF ("local outlier factor") algorithm, which identifies density-based local outliers.

The algorithm we are going to use (from the package DDoutlier) computes a local density for observations with a given k-nearest neighbors (we choose k = 5). This local density is compared to the density of the respective nearest neighbors, resulting in the local outlier factor.

Therefore, the function returns a vector of LOF scores for each observation: the greater the LOF, the greater the outlierness of the data point.

```
#install.packages('DDoutlier')
library("DDoutlier")
lof <- LOF(Prostate_reduced, k = 5) # outlier score with a neighborhood of 5 points
```

We can show the lof scores for the 5 first observations:

```
head(lof)
```

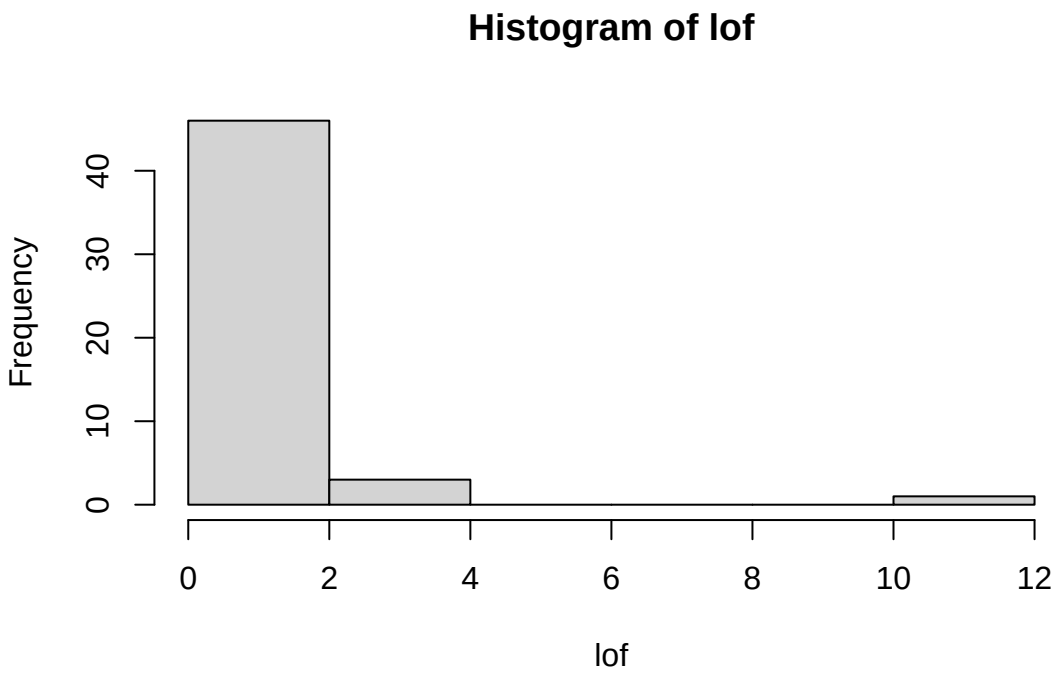
```
## [1] 2.937176 1.076620 1.119548 1.087195 1.027537 1.254277
```

We can see and visualize the distribution of outlier scores:

```
summary(lof) # some statistics
```

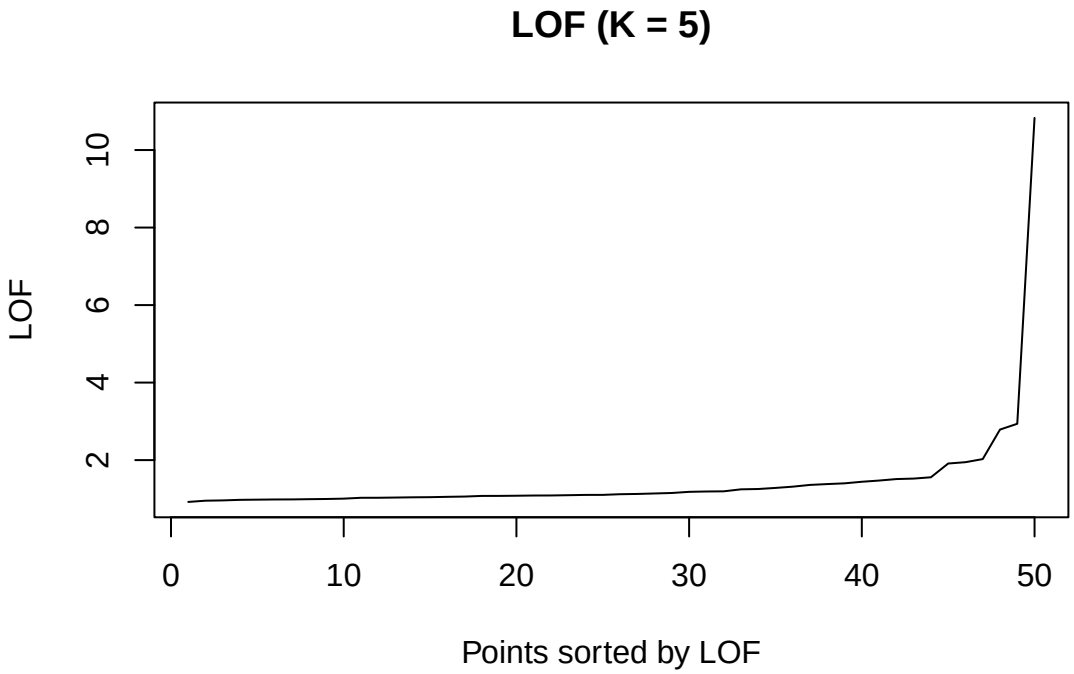
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.921   1.035   1.111   1.460   1.376  10.829
```

```
hist(lof)
```



It could be useful to plot also the sorted LOF scores:

```
plot(sort(lof), type = "l",  main = "LOF (K = 5)",
  xlab = "Points sorted by LOF", ylab = "LOF")
```



Looks like outliers start around a LOF value of 2.0.

Let's show the indexes for 5 most outlying observations

```
lof_with_names = lof
names(lof_with_names) <- 1:nrow(Prostate)
sort(lof_with_names, decreasing = TRUE)[1:5]
```

```
##        13         1        47        32        22
## 10.829292  2.937176  2.788041  2.025857  1.946327
```

Let's first find the indexes of the outliers with a lof score above 2.0:

```
outliers_lof <- which(lof > 2.0)
```

Number of detected outliers:

```
length(outliers_lof)
```

```
## [1] 4
```

```
outliers_lof
```

```
## [1]  1 13 32 47
```

We will simply remove the found outliers (found with lof) from the dataset (taking into account that k-means algorithm is very sensible to outliers):

```
Prostate_no_outliers = Prostate_reduced[-outliers_lof,]
```

Let's now check again the dimensions:

```
dim(Prostate_no_outliers)
```

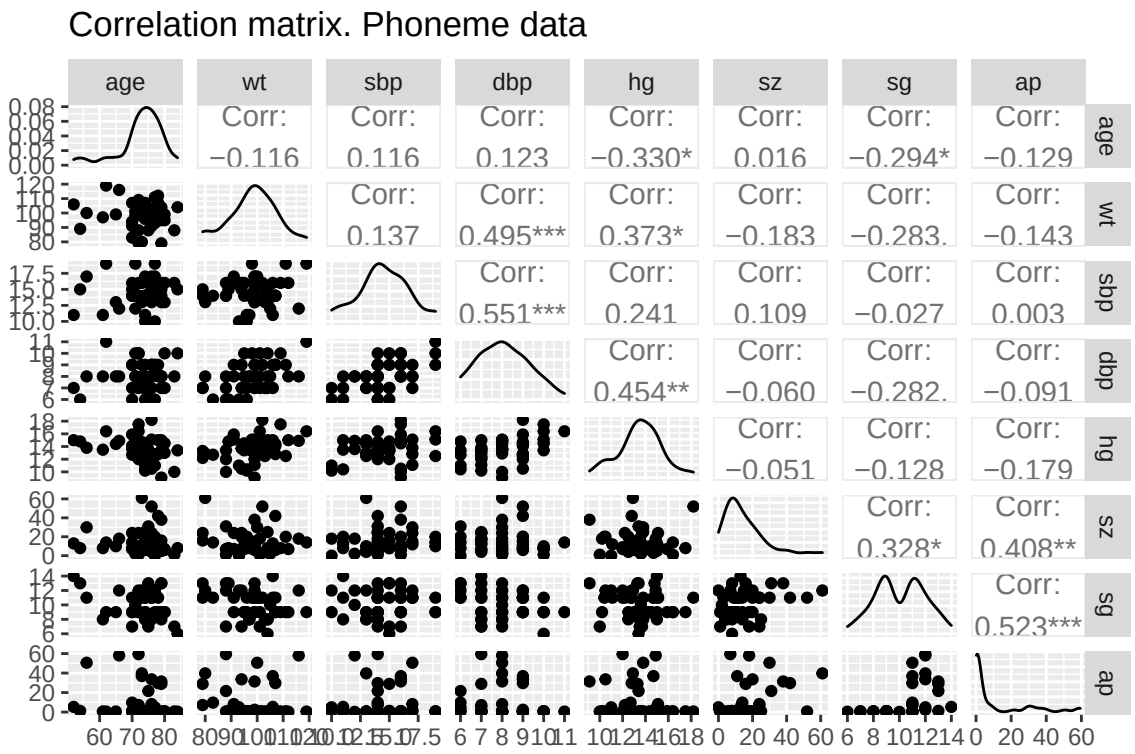
```
## [1] 46  8
```

The outliers have been correctly removed!

1.0.3 Check variables distribution

Check variables correlation:

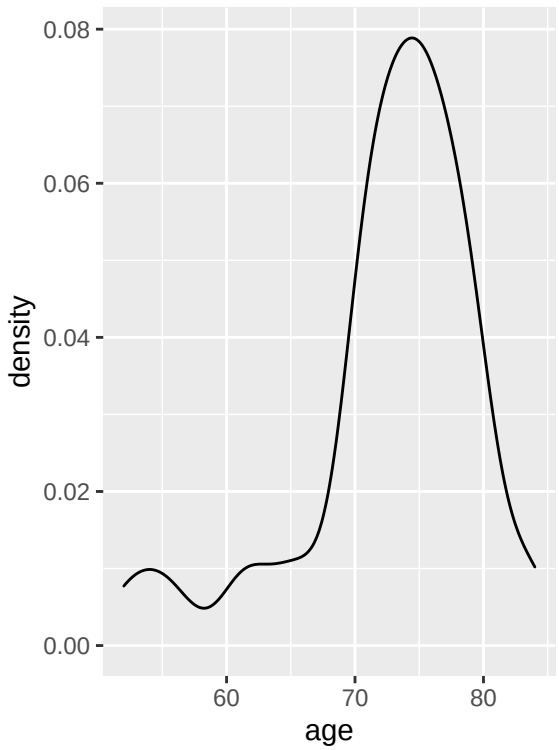
```
library(GGally)
ggpairs(Prostate_no_outliers,
        title="Correlation matrix. Phoneme data")
```

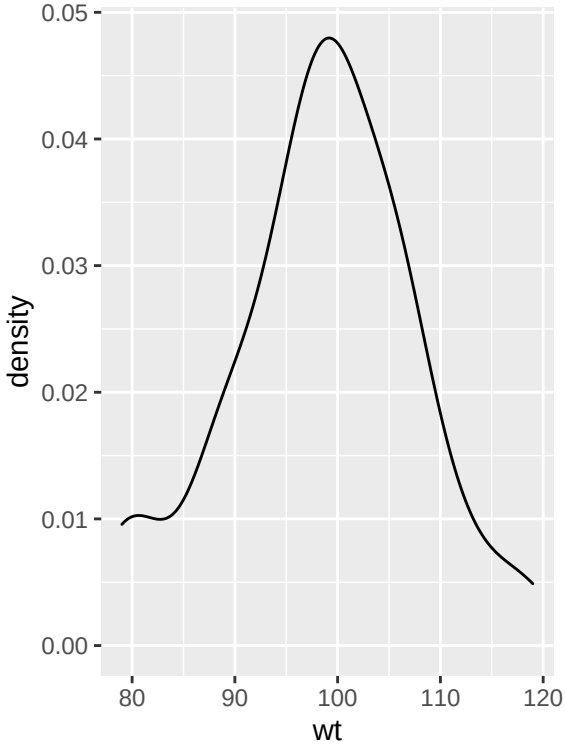


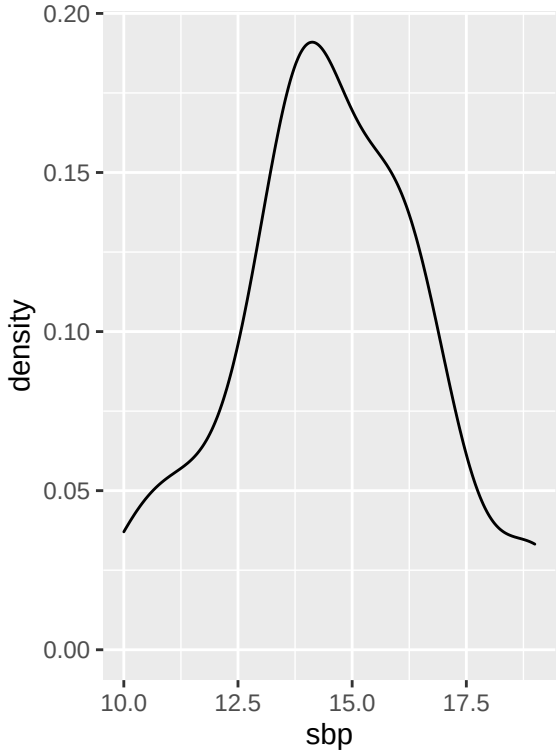
Let's now check the univariate distribution of each variable.

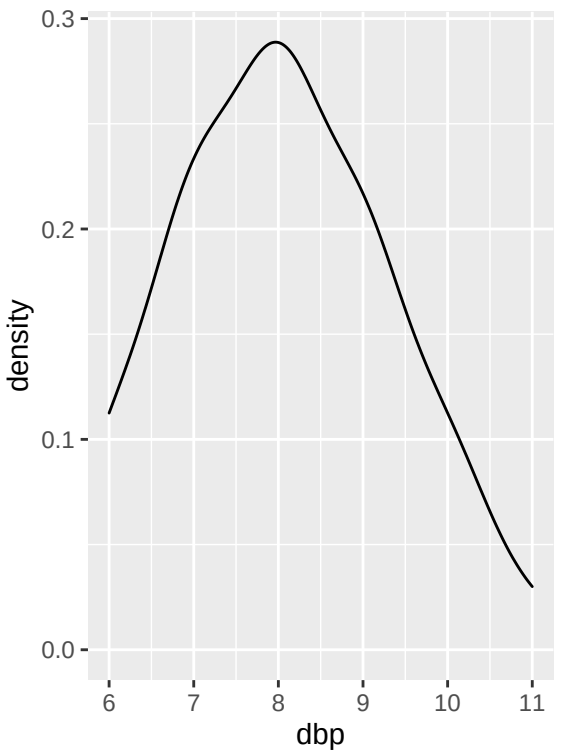
We'll use the density plot:

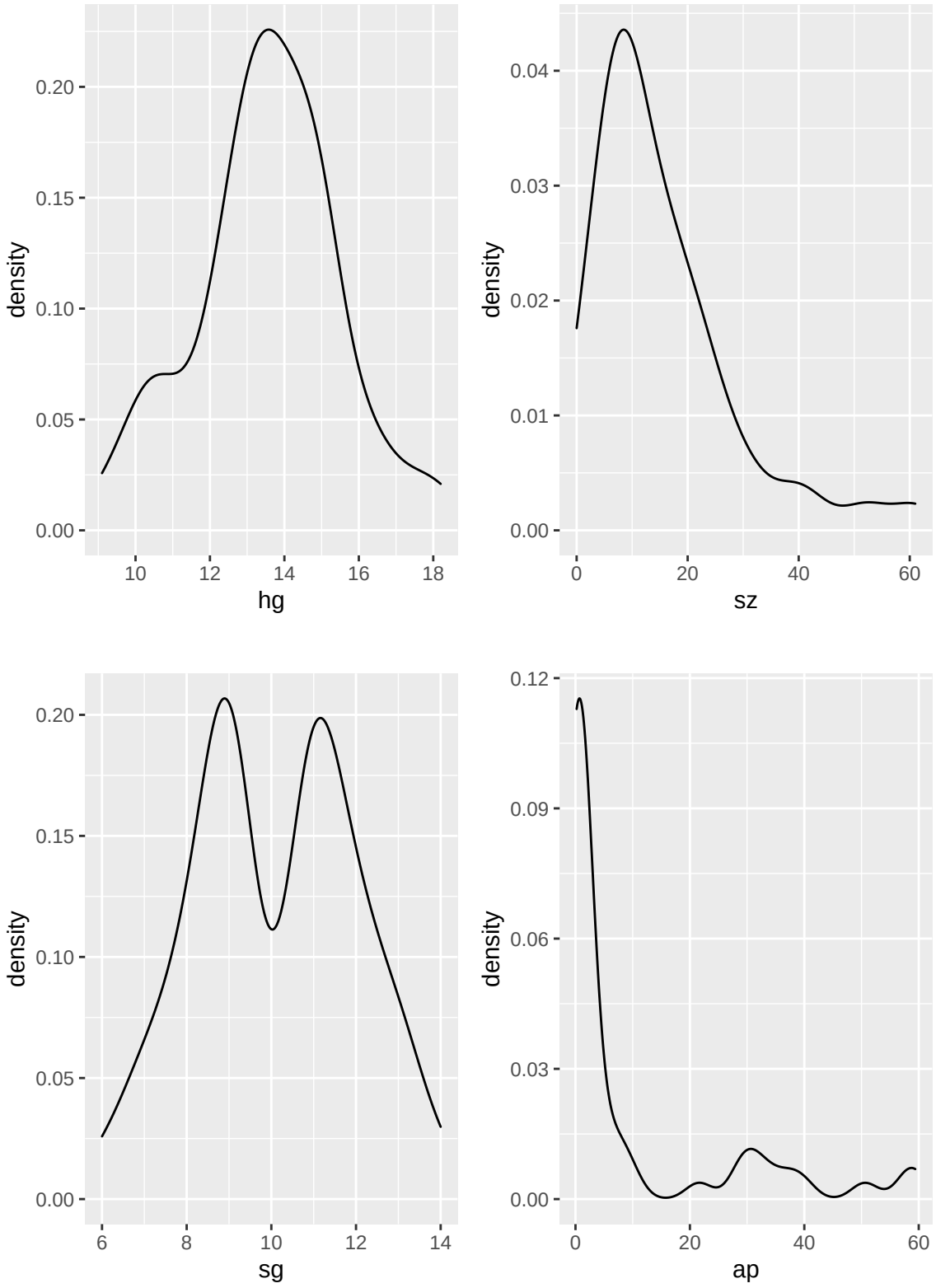
```
library(gridExtra)
library(ggplot2)
g1 <- ggplot(Prostate_no_outliers, aes(x=age)) + geom_density(alpha=0.8)
g2 <- ggplot(Prostate_no_outliers, aes(x=wt)) + geom_density(alpha=0.8)
g3 <- ggplot(Prostate_no_outliers, aes(x=sbp)) + geom_density(alpha=0.8)
g4 <- ggplot(Prostate_no_outliers, aes(x=dbp)) + geom_density(alpha=0.8)
g5 <- ggplot(Prostate_no_outliers, aes(x=hg)) + geom_density(alpha=0.8)
g6 <- ggplot(Prostate_no_outliers, aes(x=sz)) + geom_density(alpha=0.8)
g7 <- ggplot(Prostate_no_outliers, aes(x=sg)) + geom_density(alpha=0.8)
g8 <- ggplot(Prostate_no_outliers, aes(x=ap)) + geom_density(alpha=0.8)

grid.arrange(g1,g2, nrow=1); grid.arrange(g3,g4,nrow=1);grid.arrange(g5,g6,nrow=1); grid.arrange(g7,g8,nrow=1)
```


density
0.08
0.06
0.04
0.02
0.00
60
70
80
age


density
0.05
0.04
0.03
0.02
0.01
0.00
80
90
100
110
120
wt


density
0.20
0.15
0.10
0.05
0.00
10.0
12.5
15.0
17.5
sbp

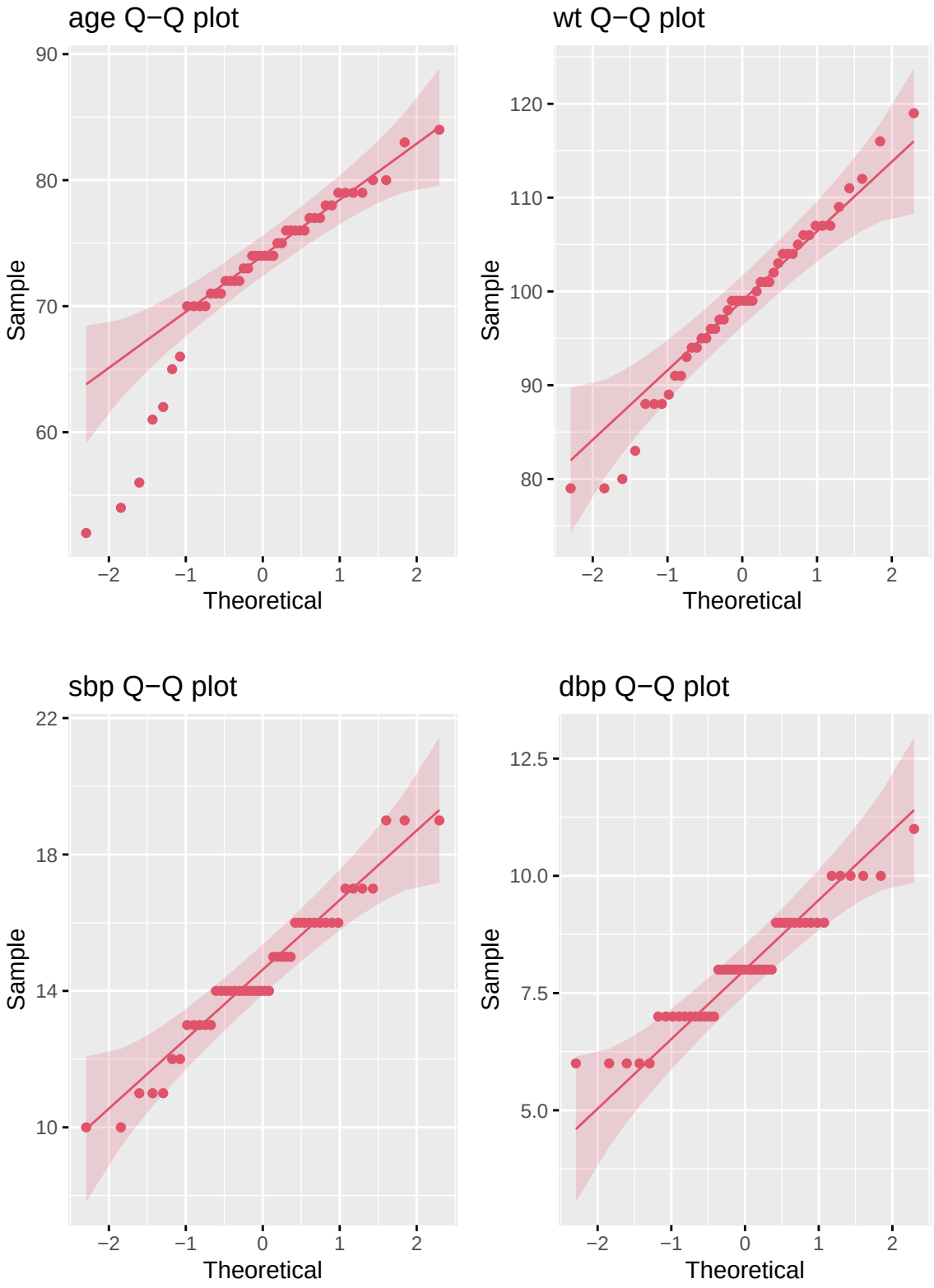

density
0.3
0.2
0.1
0.0
6
7
8
9
10
11
dbp

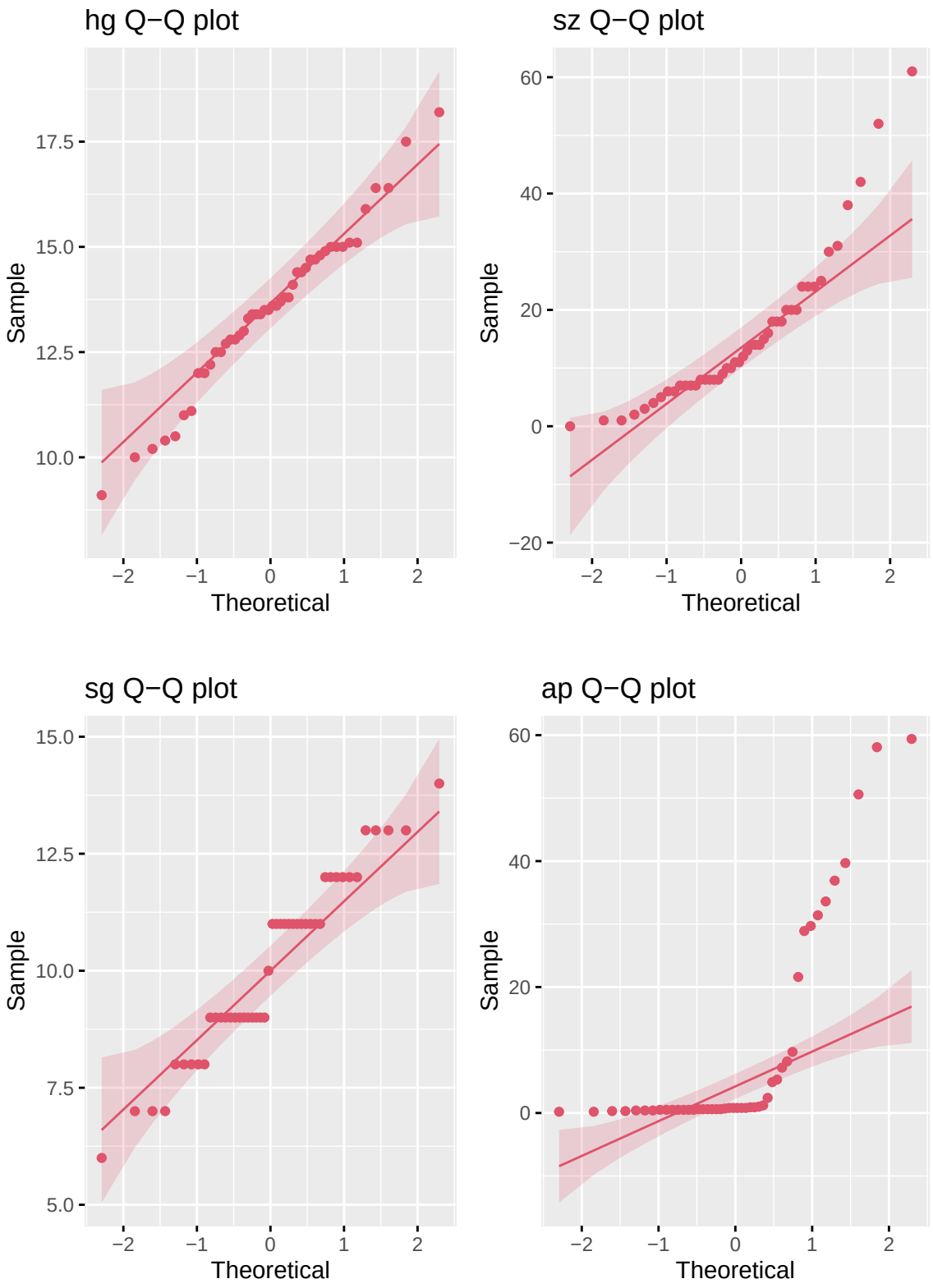


A better approach for a visual inspection is the *Q-Q plot*, which shows the distribution of the data against the expected normal distribution. In particular, for normally distributed data, observations should lie approximately on a straight line. If the data is non-normal, the points form a curve that deviates markedly from a straight line. Let's perform such plot for each predictor, by using the library *ggpubr*:

```
library(ggpubr)

g1 <- ggqqplot(Prostate_no_outliers, x="age", col=2, ggtheme = theme_gray(), title = "age Q-Q plot")
g2 <- ggqqplot(Prostate_no_outliers, x="wt", col=2, ggtheme = theme_gray(), title = "wt Q-Q plot")
```

```
g3 <- ggqqplot(Prostate_no_outliers, x="sbp", col=2, ggtheme = theme_gray(), title = "sbp Q-Q plot")
g4 <- ggqqplot(Prostate_no_outliers, x="dbp", col=2, ggtheme = theme_gray(), title = "dbp Q-Q plot")
g5 <- ggqqplot(Prostate_no_outliers, x="hg", col=2, ggtheme = theme_gray(), title = "hg Q-Q plot")
g6 <- ggqqplot(Prostate_no_outliers, x="sz", col=2, ggtheme = theme_gray(), title = "sz Q-Q plot")
g7 <- ggqqplot(Prostate_no_outliers, x="sg", col=2, ggtheme = theme_gray(), title = "sg Q-Q plot")
g8 <- ggqqplot(Prostate_no_outliers, x="ap", col=2, ggtheme = theme_gray(), title = "ap Q-Q plot")
grid.arrange(g1,g2, nrow=1); grid.arrange(g3,g4,nrow=1);grid.arrange(g5,g6,nrow=1);grid.arrange(g7,g8,nrow=1)
```





The plots above clearly show that most of the variables does not follow a normal distribution.

To have more precise insights, we can apply the normality Shapiro-Wilk normality test to each variable:

```
print(shapiro.test(Prostate_no_outliers$age))
```

```
##
##  Shapiro-Wilk normality test
##
```

```
## data:  Prostate_no_outliers$age
## W = 0.88252, p-value = 0.0002428
```

```
print(shapiro.test(Prostate_no_outliers$wt))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Prostate_no_outliers$wt
## W = 0.9802, p-value = 0.6142
```

```
print(shapiro.test(Prostate_no_outliers$sbp))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Prostate_no_outliers$sbp
## W = 0.95818, p-value = 0.09737
```

```
print(shapiro.test(Prostate_no_outliers$dbp))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Prostate_no_outliers$dbp
## W = 0.93459, p-value = 0.01236
```

```
print(shapiro.test(Prostate_no_outliers$hg))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Prostate_no_outliers$hg
## W = 0.98095, p-value = 0.645
```

```
print(shapiro.test(Prostate_no_outliers$sz))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Prostate_no_outliers$sz
## W = 0.84591, p-value = 2.41e-05
```

```
print(shapiro.test(Prostate_no_outliers$sg))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Prostate_no_outliers$sg
## W = 0.94809, p-value = 0.0396
```

```
print(shapiro.test(Prostate_no_outliers$ap))
```

Table 3: Prostate cancer data set (sample of 20)

age	wt	sbp	dbp	hg	sz	sg	ap
0.6149526	0.05768309	1.1431953	0.76740036	-0.3853005	-0.03343536	1.0031600	-0.4343074
0.3248211	0.27880162	-0.2406727	-0.83716403	-0.0200182	0.19726860	-1.6187355	-0.5475538
-0.2554419	-0.05287617	2.0657739	1.56968255	0.8133139	-0.41794194	0.4787809	-0.5415946
-1.5610336	2.26886836	2.0657739	2.37196474	1.4899510	-0.11033667	-0.5699773	-0.5475538
0.4698869	0.49992015	-0.7019620	-0.83716403	-0.5409779	-0.87934985	-1.0943564	-0.5475538
-0.9807706	1.93719057	-1.1632513	-0.03488183	0.7085114	0.19726860	1.0031600	2.8854044
0.1797554	-0.27399470	-2.0858299	-0.83716403	-1.7395923	0.19726860	-0.5699773	-0.5237170
0.4698869	-0.49511322	0.6819059	0.76740036	-1.5828974	0.35107123	1.0031600	1.4250467
-0.4005076	0.94215720	-0.2406727	0.76740036	0.2923541	-0.57174458	-0.5699773	-0.5654368
0.6149526	1.38439426	2.0657739	0.76740036	-0.5409779	0.35107123	-0.5699773	-0.5296762

```
##
##  Shapiro-Wilk normality test
##
## data:  Prostate_no_outliers$ap
## W = 0.61867, p-value = 1.068e-09
```

The low p-values (<0.05) for the variables 'age', 'dbp', 'sz', 'ap', 'sg' reject the null hypotheses as we expected from the plots.

However, clustering techniques related to **distance-based methods** (e.g. hierarchical clustering & k-means) **has nothing to do with whether the variables** belong to some known distribution such as the **normal distribution**. For this reason we do not apply any transformation to change variables distributions.

1.0.4 Scaling

The **units** of the variables might have **an influence** in the clustering results for what regards clustering methods based on distances such as k-means and the hierarchical one. We do not want the results of those clustering methods to depend on a variable with very large units, so we **scale (standardize) each variable**:

```
Prostate_no_outliers_sc <- Prostate_no_outliers %>%
        mutate_if(is.numeric, scale)
```

We can print a sample of the scaled table to see the new units:

```
Prostate_no_outliers_sc %>%
  sample_n(., 10, replace=FALSE) %>%
  kbl(caption = "Prostate cancer data set (sample of 20)") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

Let's print again the basic statistics:

```
Prostate_no_outliers_sc %>%
  summary(.) %>%
  kbl(caption = "Basic statistics. Prostate cancer data set") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

As we can see, we have now zero mean.

2 Hierarchical clustering

In this section we are going to apply various hierarchical clustering methods that allow to draw dendrograms, and then we'll compare their results.

The first experiment that we are going to run is to differentiate agglomerative and divisive clustering:

Table 4: Basic statistics. Prostate cancer data set

	age.V1	wt.V1	sbp.V1	dbp.V1	hg.V1	sz.V1
	Min. :-3.0116910	Min. :-2.1535022	Min. :-2.0858299	Min. :-1.6394462	Min. :-2.3124445	Min. :-1.186955
	1st Qu.:-0.2554419	1st Qu.:-0.4951132	1st Qu.:-0.5866397	1st Qu.:-0.8371640	1st Qu.:-0.5150317	1st Qu.:-0.64864
	Median : 0.1797554	Median : 0.0576831	Median :-0.2406727	Median :-0.0348818	Median : 0.0059280	Median :-0.3025
	Mean : 0.0000000	Mean : 0.0000000	Mean : 0.0000000	Mean : 0.0000000	Mean : 0.0000000	Mean : 0.000000
	3rd Qu.: 0.6149526	3rd Qu.: 0.6104794	3rd Qu.: 0.6819059	3rd Qu.: 0.7674004	3rd Qu.: 0.6436458	3rd Qu.: 0.3510
	Max. : 1.6304128	Max. : 2.2688684	Max. : 2.0657739	Max. : 2.3719647	Max. : 2.4280856	Max. : 3.504025

- Agglomerative Clustering: also known as a bottom-up approach, starts with singletons and then agglomerates them based on their distances until having a group containing the whole set of elements.
- Divisive Clustering: also known as a top-down approach, starts with one group (or cluster) containing every element, then divides them using a splitting method until reaching singletons clusters.

An important parameter of the algorithm is the distance measure to apply for calculating the distances between points. A further parameter is needed to calculate the distance between groups. We'll explore all of them in the following.

2.1 Agglomerative hierarchical clustering

To experiment the agglomerative hierarchical clustering, we'll use the *agnes* package. We'll compute agglomerative hierarchical clustering using the euclidean distance for the points, while we'll compare 4 different approaches for what regards groups distances: - Simple linkage - Average linkage - Complete linkage - Ward criterion

Since *agnes* returns also the 'agglomerative coefficient', we'll compare the solutions obtained using the various distance approaches we mentioned above using the agglomerative coefficient that represents the amount of clustering structure found (values closer to 1 suggest a strong clustering structure).

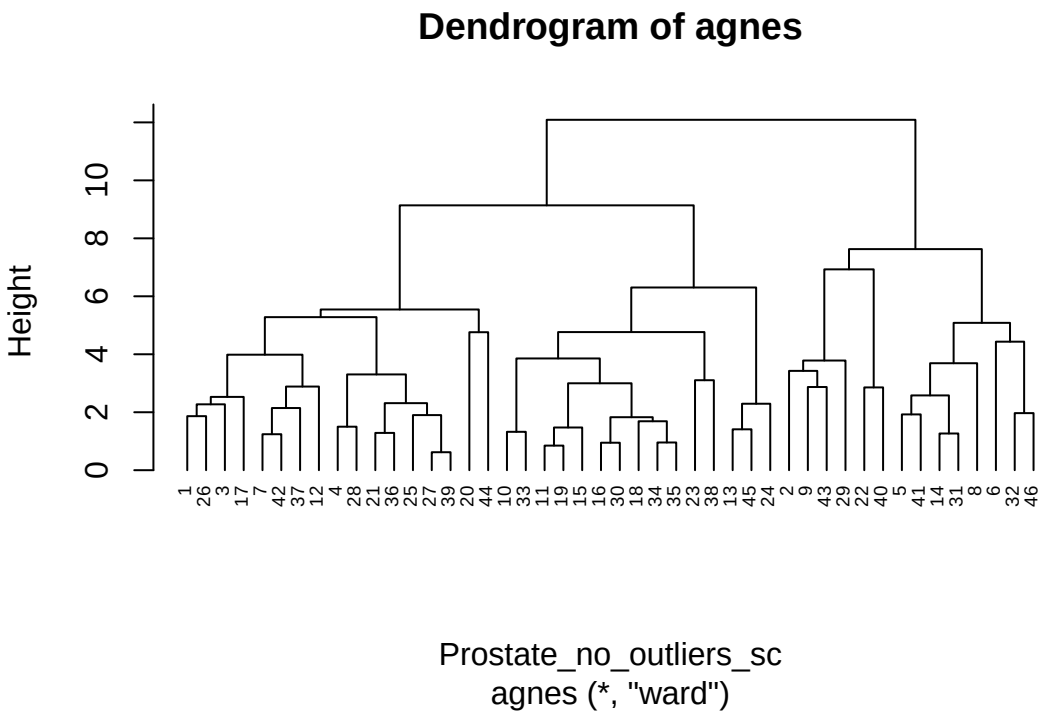
Let's compare the 4 approaches, by using the euclidean distance among points:

```
# vector of methods to compare
m <- c("average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")

# function to compute coefficient
ac <- function(x) {
  agnes(Prostate_no_outliers_sc, method = x)$ac
}
library(purrr)
map_dbl(m, ac)
```

```
##   average    single  complete      ward
## 0.6073427 0.4517574 0.7399859 0.8309064
```

Among the 4 distance computation methods, Ward's method gave us the highest agglomerative coefficient. Therefore, we can inspect its dendrogram:

```
agnes.ward <- agnes(Prostate_no_outliers_sc, method = "ward")
agnes_dend <- agnes.ward %>% as.dendrogram
pltree(agnes.ward, cex = 0.6, hang = -1, main = "Dendrogram of agnes")
```



2.1.1 Cutting the tree

In hierarchical clustering we should decide where to cut the dendrogram, in order to obtain an actual assignment of observations to clusters: this means that we should decide the optimal number of clusters.

We can do that with the function *cutree* which cuts a tree into several groups either by specifying the desired number of groups or the cut height.

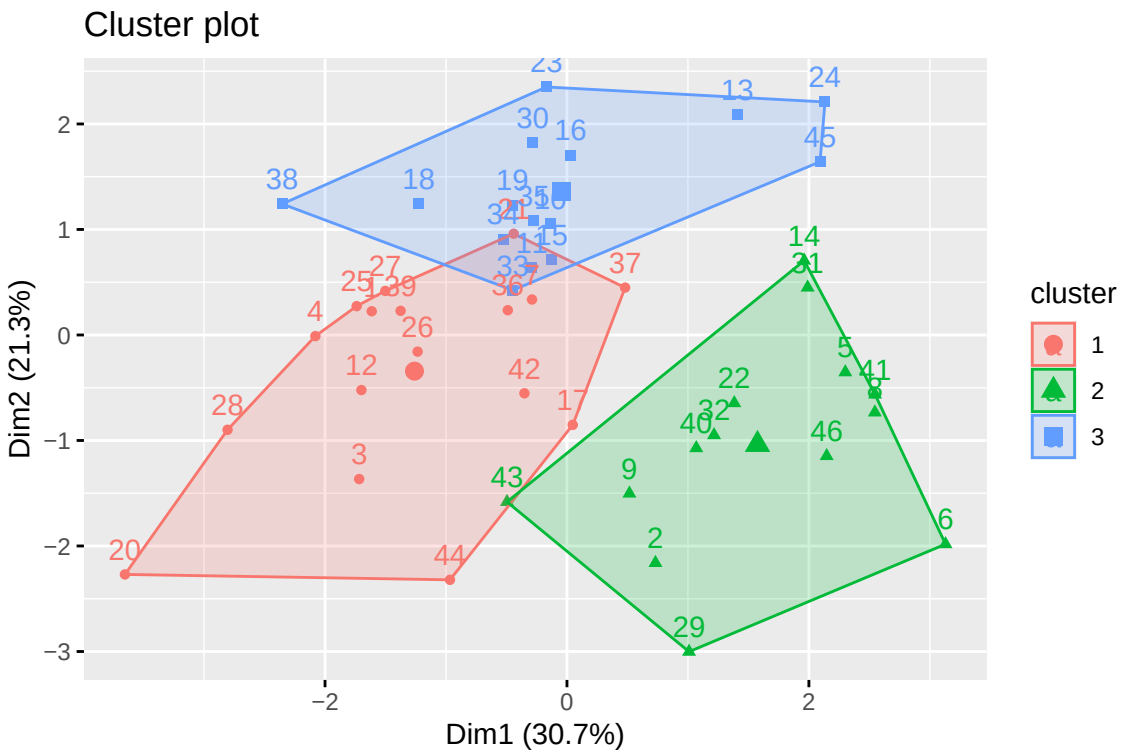
One typical approach consists in cutting the tree where the branches are quite long: in the above dendrogram we can see that this happens at a height between 8 and 9, where we have 3 clusters. Let's then perform the cutting:

```
(clust <- cutree(agnes.ward, k = 3))
```

```
##  [1] 1 2 1 1 2 2 1 2 2 3 3 1 3 2 3 3 1 3 3 1 1 2 3 3 1 1 1 1 2 3 2 2 3 3 3 1 1 3
## [39] 1 2 2 1 2 1 3 2
```

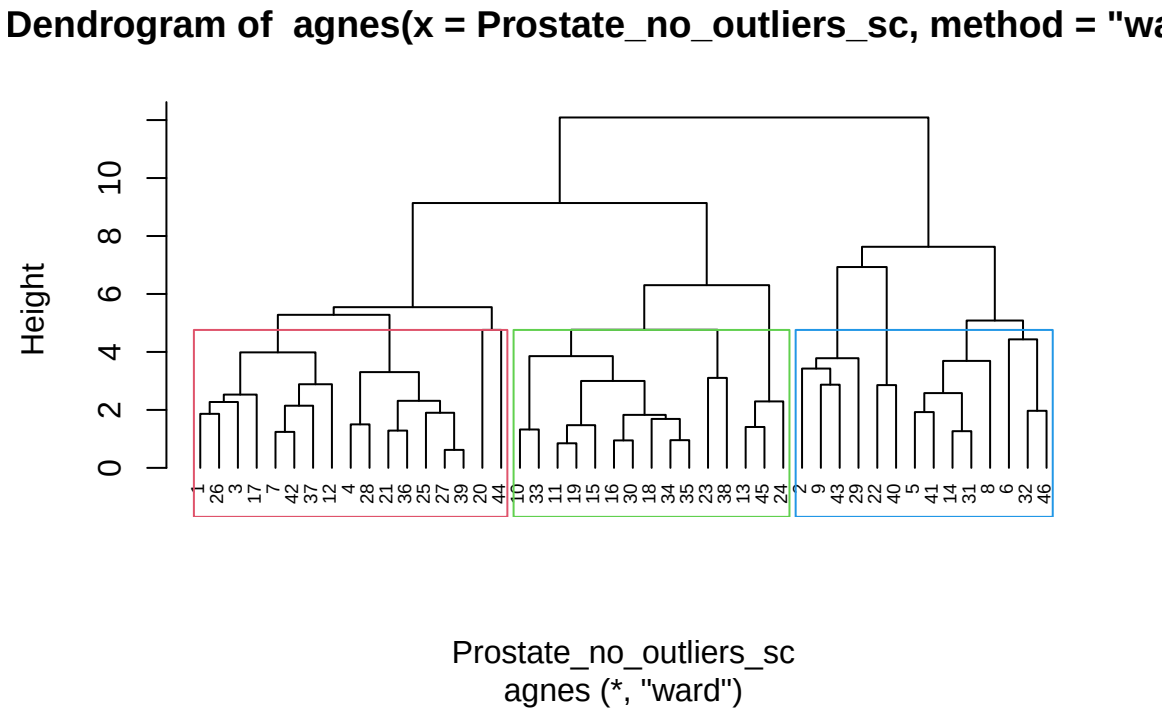
In order to graph those clusters, we can use the function *fviz_cluster* from the package *factorextra*:

```
fviz_cluster(list(data = Prostate_no_outliers_sc, cluster = clust))
```



We can also depict the clusters within the dendrogram in this way:

```
pltree(agnes.ward, hang=-1, cex = 0.6)
rect.hclust(agnes.ward, k = 3, border = 2:5)
```



2.2 Divisive hierarchical clustering

After having performed hierarchical clustering using the agglomerative approach, it makes sense to apply the divisive option: To do so we use the *diana* algorithm. The function *diana* works in a similar way to agnes with 2 small differences: there is

no method argument here, and instead of the agglomerative coefficient, we have a division coefficient.

```
library(dendextend)
diana_dend <- diana(Prostate_no_outliers_sc) %>% as.dendrogram
diana_dend <- color_branches(diana_dend, k = 2)
```

We proceed now in computing the divisive coefficient: if it is closer to one suggests stronger group distinctions. The divisive coefficient is:

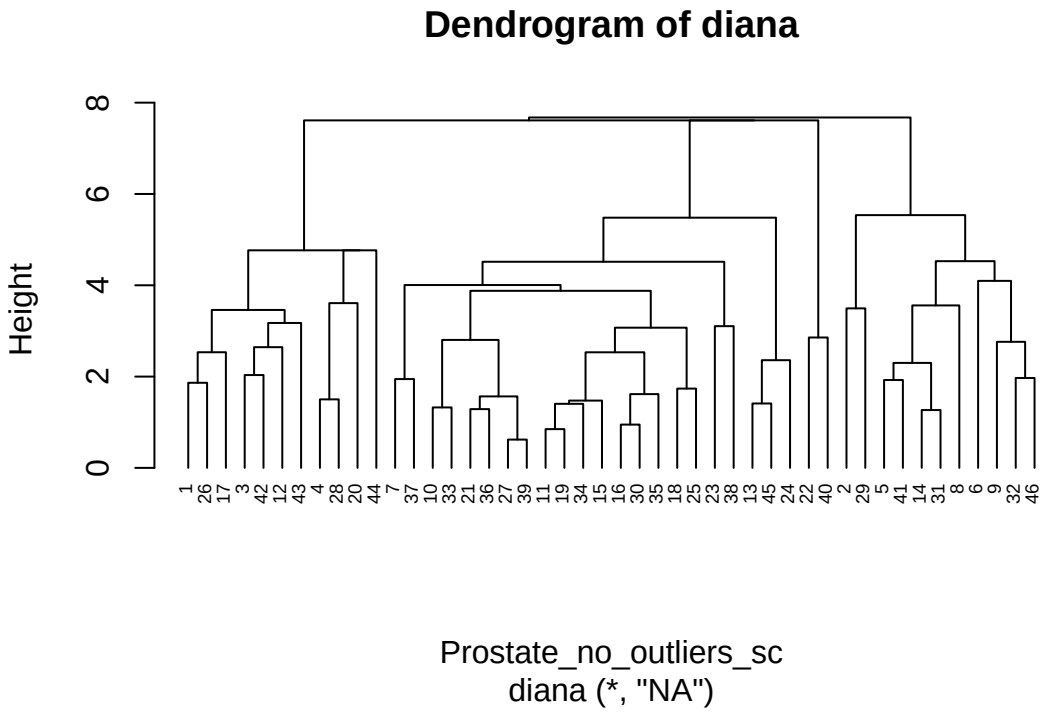
```
diana.hc <- diana(Prostate_no_outliers_sc)
diana.hc$dc
```

```
## [1] 0.7330041
```

We obtain a value slightly lower than the agglomerative coefficient obtained with the agglomerative approach (0.8309064).

Finally, we plot of the dendogram:

```
pltree(diana.hc, cex = 0.6, hang = -1, main = "Dendrogram of diana")
```



2.2.1 Cutting the tree

By looking at the length of the branches, the above dendrogram suggests to cut at a height around 6, which gives us 4 clusters.

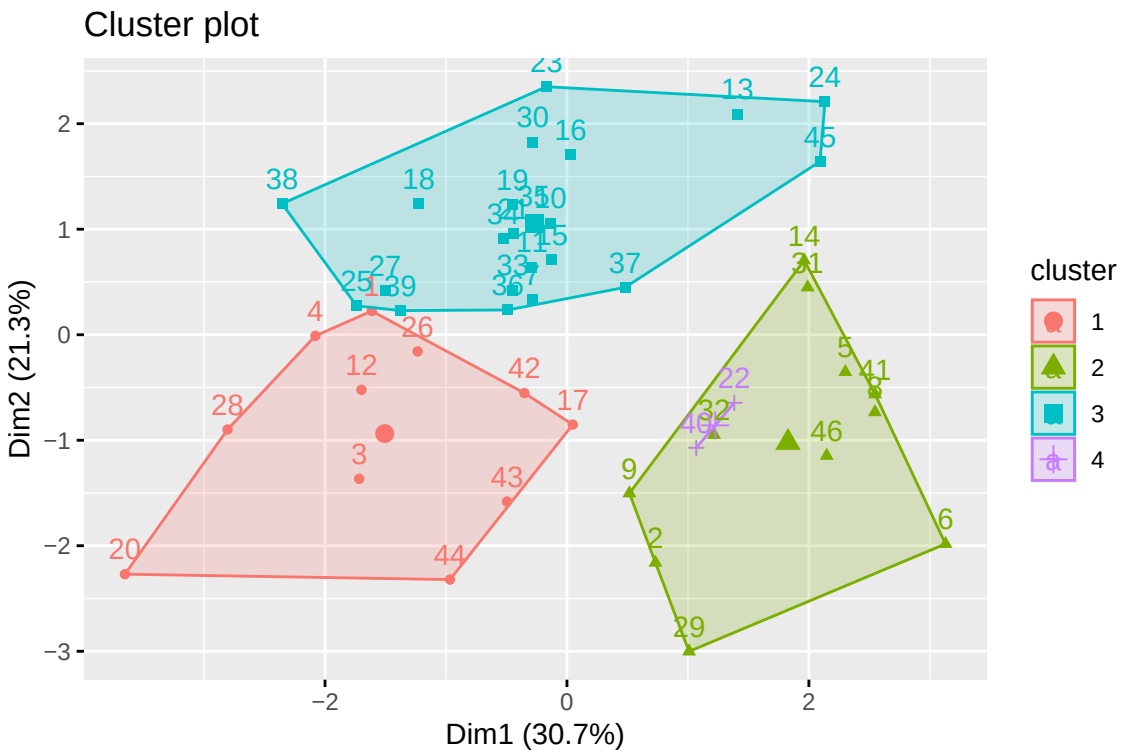
Let's then perform the cutting:

```
(clust <- cutree(diana_dend, k = 4))
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  1  2  1  1  2  2  3  2  2  3  3  1  3  2  3  3  1  3  3  1  3  4  3  3  3  1
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
##  3  1  2  3  2  2  3  3  3  3  3  3  3  4  2  1  1  1  3  2
```

We can graph the clusters:

```
fviz_cluster(list(data = Prostate_no_outliers_sc, cluster = clust))
```



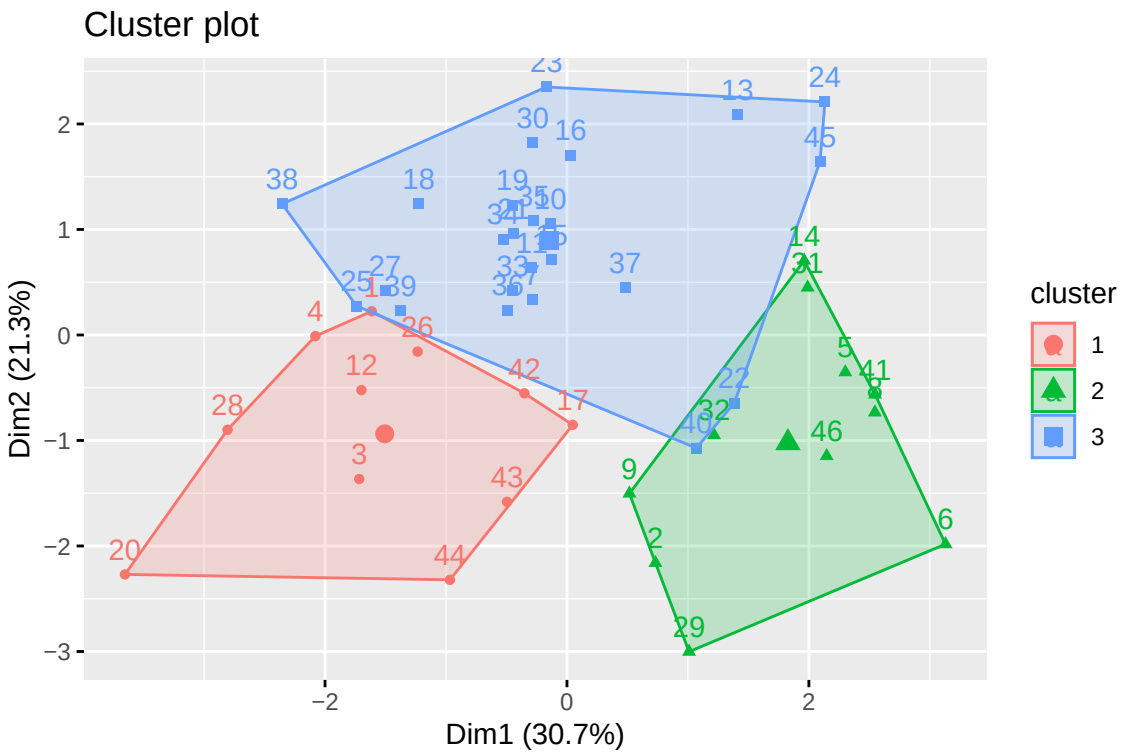
It seems that the 4th cluster is not well representative of a class of individuals: indeed it is formed only by 2 observations. We therefore re-perform the cutting with k=3:

```
(clust <- cutree(diana_dend, k = 3))
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  1  2  1  1  2  2  3  2  2  3  3  1  3  2  3  3  1  3  3  1  3  3  3  3  3  1
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
##  3  1  2  3  2  2  3  3  3  3  3  3  3  3  2  1  1  1  3  2
```

We can graph again the clusters:

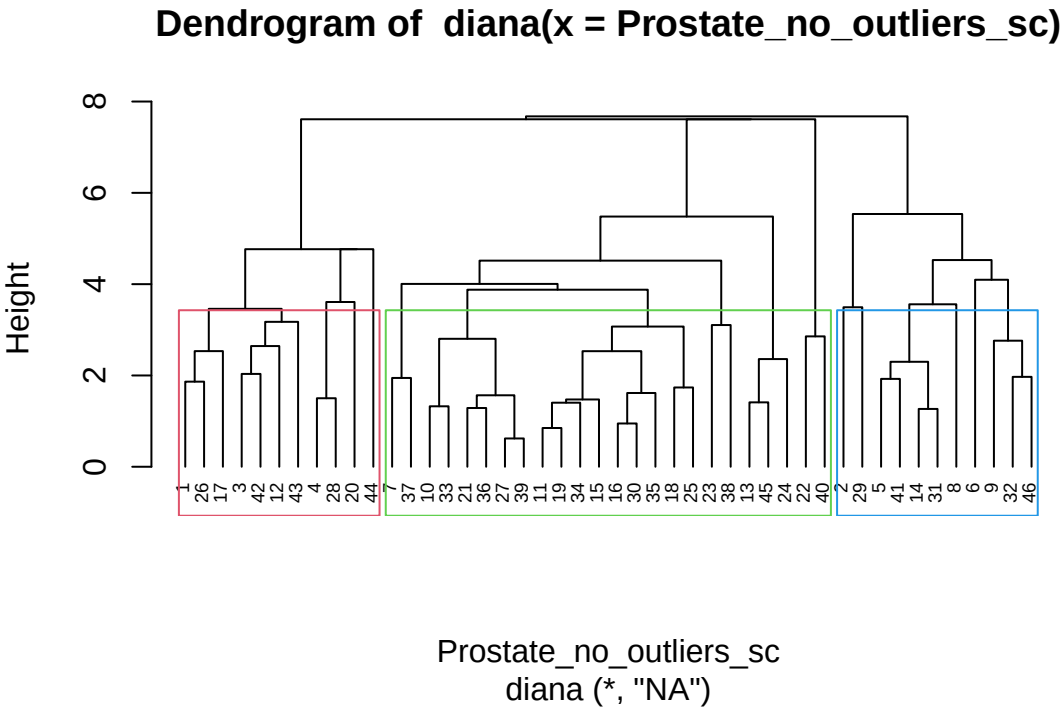
```
fviz_cluster(list(data = Prostate_no_outliers_sc, cluster = clust))
```



The plots seems now much more consistent, with a small overlapping between clusters 2 and 3.

We can also depict the clusters within the dendrogram in this way:

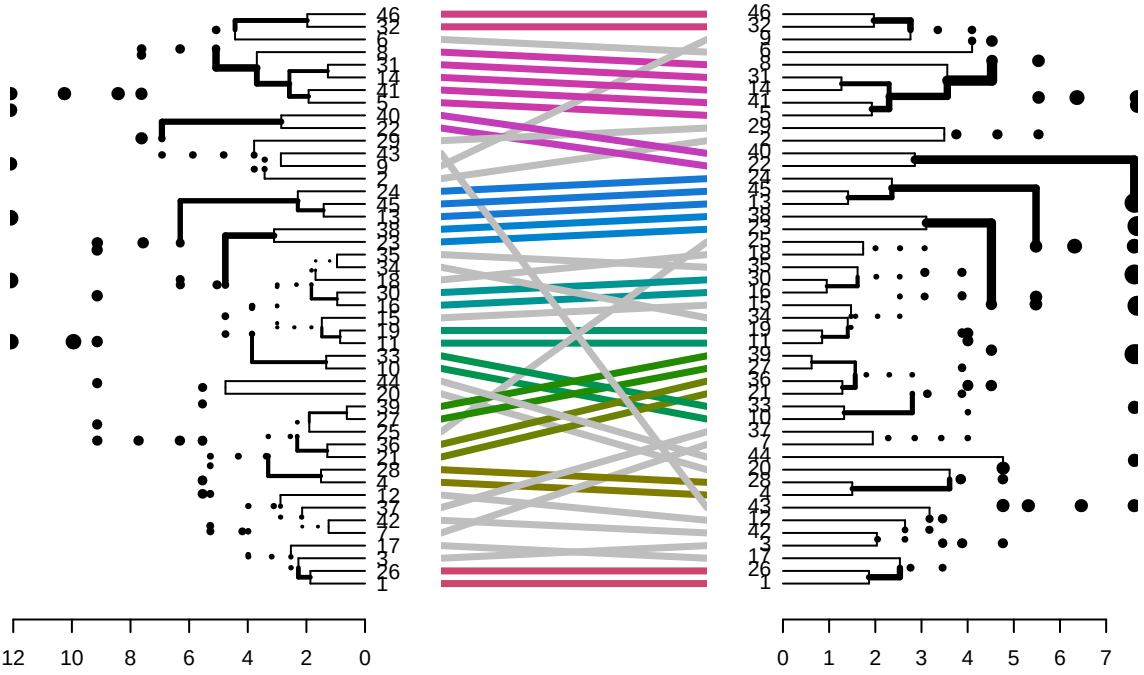
```
pltree(diana.hc, hang=-1, cex = 0.6)
rect.hclust(diana.hc, k = 3, border = 2:5)
```



2.3 Comparing agglomerative and divisive methods

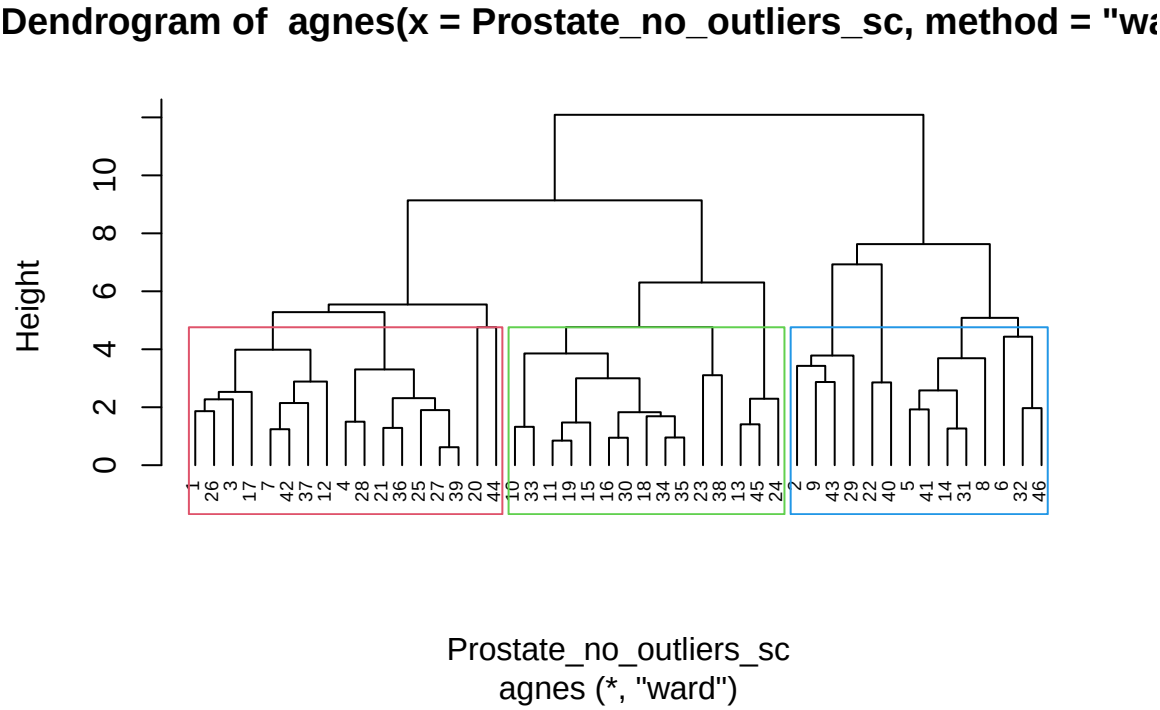
We can compare the partitions obtained throught the two approaches by using a *tanglegram* which makes us able to compare the two dendograms by drawing links between them:

```
library(dendextend)
diana_dend <- diana.hc %>% as.dendrogram
tanglegram(agnes_dend, diana_dend)
```

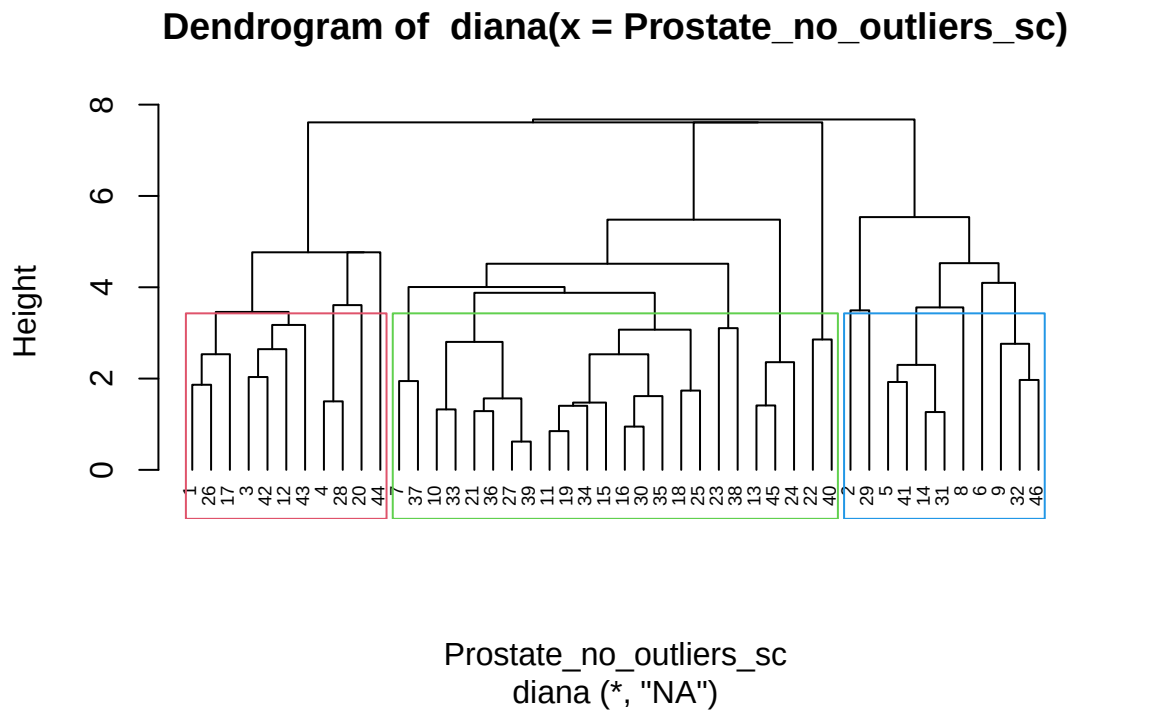


A better way to visualize the differences among the partitions obtained with the 2 algorithms, is to visualize the 2 dendrograms one below the other:

```
pltree(agnes.ward, hang=-1, cex = 0.6)
rect.hclust(agnes.ward, k = 3, border = 2:5)
```




```
pltree(diana.hc, hang=-1, cex = 0.6)
rect.hclust(diana.hc, k = 3, border = 2:5)
```



The 3 clusters are slighlty different between the methods: for instance we can see that the 'green' cluster is bigger in 'diana' with respect to 'agnes'.

2.4 Hierarchical clustering on principal components using *HCPC*

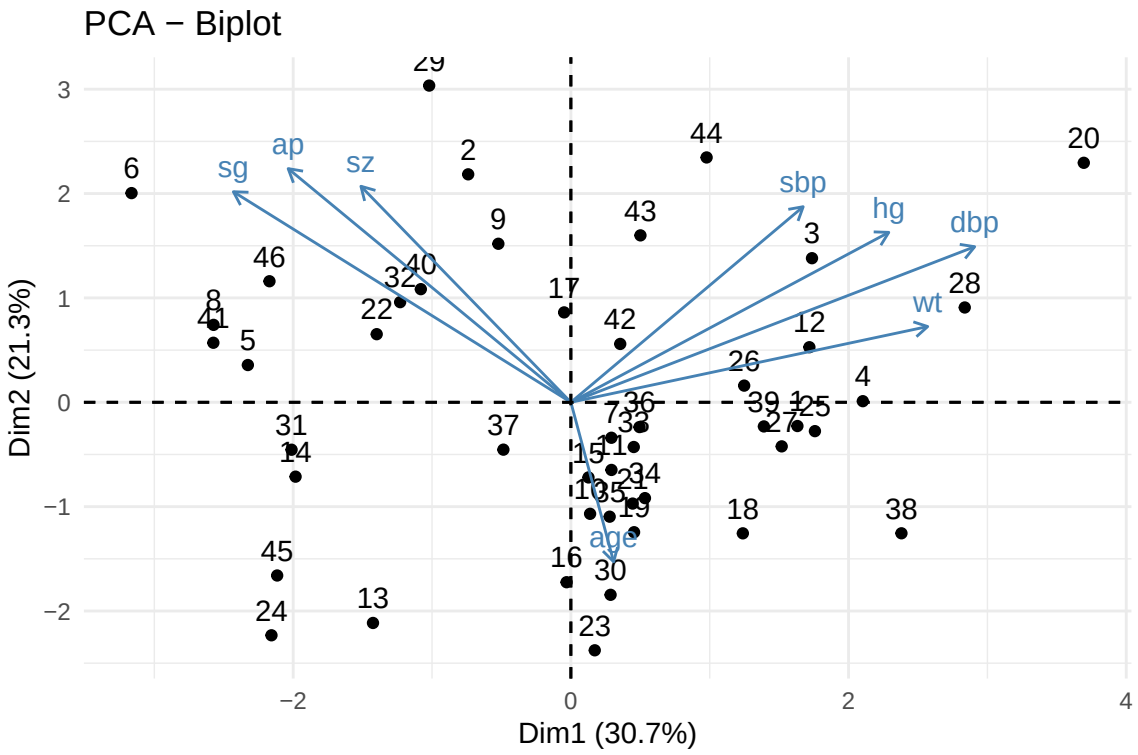
Another approach consists in applying firstly a PCA to the continuous dataset, and then a clustering method to the obtained reduced representation.

In this approach, the PCA step can be considered as a denoising step which can lead to a more stable clustering. This might be useful in our case since we are doing clustering by considering 8 variables.

The algorithm of the HCPC method, as implemented in the *FactoMineR* package, can be summarized as follow:

1. *Compute PCA*: we retain 3 dimensions

```
res.pca <- PCA(Prostate_no_outliers_sc, ncp = 3, graph = FALSE)
fviz_pca_biplot(res.pca) +
       theme_minimal()
```



2. *Compute hierarchical clustering*: Hierarchical clustering is performed using the Ward's criterion on the selected principal components. Ward criterion is used in the hierarchical clustering because it is based on the multidimensional variance like principal component analysis.

```
res.hcpc <- HCPC(res.pca, graph = FALSE)
```

The function *HCPC* returns a list containing:

- *data.clust*: The original data with a supplementary column called class containing the partition.
- *desc.var*: The variables describing clusters
- *desc.ind*: The more typical individuals of each cluster
- *desc.axes*: The axes describing clusters

Therefore, the *HCPC* function already performs an initial partitioning. To know the number of clusters:

```
unique(res.hcpc$data.clust$clust)
```

Table 5: Cluster allocation (sample of 10)

age	wt	sbp	dbp	hg	sz	sg	ap	clust
0.9050841	0.61047941	-0.7019620	-0.03488183	0.76142140	-0.8024485	-1.0943564	-0.5475538	2
0.1797554	-0.16343543	-0.2406727	-0.03488183	0.03187428	-0.1103367	-1.0943564	-0.5475538	2
0.1797554	-0.27399470	-2.0858299	-0.83716403	-1.73959232	0.1972686	-0.5699773	-0.5237170	2
-2.4314280	0.16824236	1.1431953	-0.03488183	0.13565922	1.1200844	0.4787809	2.4383561	1
1.6304128	0.61047941	0.2206166	1.56968255	-0.07292817	-0.5717446	-2.1431146	-0.5535166	3
-0.1103761	1.16327573	0.6819059	1.56968255	2.06382074	-0.5717446	-0.5699773	-0.5356354	3
1.4853471	-1.15846880	0.6819059	-0.03488183	-1.84337727	-1.1100538	-1.6187355	-0.5415946	2
0.1797554	0.05768309	1.1431953	-0.83716403	-1.27052506	-0.2641393	0.4787809	-0.5535166	2
0.9050841	0.05768309	0.6819059	-0.03488183	-2.31244453	1.7352949	1.5275391	1.2941920	1
0.1797554	-0.27399470	-1.6245406	-1.63944622	-1.63580737	0.3510712	0.4787809	-0.5415946	2

```
## [1] 3 1 2
## Levels: 1 2 3
```

The method returns the same number of clusters we found with the previous approaches (3). Thus, let's display the original data with cluster assignments:

```
res.hcpc$data.clust %>%
  sample_n(., 10, replace=FALSE) %>%
  kbl(caption = "Cluster allocation (sample of 10)") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

In the table above, the last column contains the cluster assignments.

To display quantitative variables that describe the most each cluster:

```
res.hcpc$desc.var$quanti
```

```
## $`1`
##        v.test Mean in category  Overall mean sd in category Overall sd
## ap   4.888315        1.1483270 -3.848446e-17      1.1231498  0.9890707
## sg   4.613774        1.0838337 -1.909704e-16      0.4973061  0.9890707
## sz   3.055738        0.7178314 -3.273348e-17      1.1382969  0.9890707
## age -2.084940       -0.4897788  6.305343e-17      1.3289875  0.9890707
## dbp -2.250172       -0.5285940  5.514912e-16      0.8046523  0.9890707
## wt  -2.614491       -0.6141770 -1.301797e-16      1.2668653  0.9890707
##          p.value
## ap  1.017026e-06
## sg  3.954232e-06
## sz  2.245077e-03
## age 3.707472e-02
## dbp 2.443800e-02
## wt  8.936043e-03
##
## $`2`
##        v.test Mean in category  Overall mean sd in category Overall sd
## sz  -2.040427       -0.3752190 -3.273348e-17      0.5846648  0.9890707
## hg  -2.250792       -0.4139036  2.359224e-16      0.8018330  0.9890707
## dbp -2.613453       -0.4805942  5.514912e-16      0.6111300  0.9890707
## ap  -2.876753       -0.5290132 -3.848446e-17      0.0599253  0.9890707
## sg  -2.941095       -0.5408451 -1.909704e-16      0.8099582  0.9890707
## sbp -3.120444       -0.5738261 -2.606611e-16      0.8671688  0.9890707
##         p.value
## sz  0.041307862
## hg  0.024398734
```

```
## dbp 0.008963249
## ap  0.004017894
## sg  0.003270542
## sbp 0.001805785
##
## $`3`
##       v.test Mean in category  Overall mean sd in category Overall sd
## dbp 4.882157        1.0348278  5.514912e-16      0.6328487  0.9890707
## sbp 3.797471        0.8049164 -2.606611e-16      0.7620199  0.9890707
## hg  3.754202        0.7957451  2.359224e-16      0.8107734  0.9890707
## wt  2.601960        0.5515145 -1.301797e-16      0.8437989  0.9890707
##          p.value
## dbp 1.049315e-06
## sbp 1.461798e-04
## hg  1.738945e-04
## wt  9.269267e-03
```

Let's focus on columns "Mean in category" (the average within the corresponding group), "Overall Mean" (the overall mean in the data set), "p.value" (<0.05 means that there is significant difference between Overall mean and the mean within the corresponding group).

From the output above, it can be seen that:

- Cluster 1 is characterized by positive values of variables 'ap', 'sg', 'sz';
- For what regard cluster 2, we have pretty high p-values with respect to the other clusters, but still we have p-values $< 5\%$ and therefore we can draw some conclusions. What we can say is that this cluster is characterized by negative (standardized) values of each one of the variables: for instance, with respect to cluster 1, in cluster 2 the variable 'sz' has a mean in the category of -0.37, while we have a positive value of the mean for the same variable on cluster 1. The same reasoning can be applied for the other variables variables.
- Cluster 3 has, on overall positive values for the varaible 'dbp' (which has negative mean in cluster 1 and 2); it has also positive value for variable 'sbp', which is also significant for cluster 2 where it has a negative mean. Also variable 'hg' has positive value in this cluster, and it's also significant in cluster 2 with negative mean value. Varaible 'wt' has also positive value in this cluster. So, overall cluster 3 is characterize by positive mean values (standardized) of the variables 'dbp', 'sbp', 'hg', 'wt'.

To show principal dimensions that are the most associated with clusters:

```
res.hcpc$desc.axes$quanti
```

```
## $`1`
##          v.test Mean in category Overall mean sd in category Overall sd
## Dim.2  3.253380         1.007758 6.542924e-17      0.9880374   1.304195
## Dim.1 -4.711159        -1.753484 1.779977e-17      0.7792434   1.567094
##            p.value
## Dim.2 1.140409e-03
## Dim.1 2.463123e-06
##
## $`2`
##          v.test Mean in category Overall mean sd in category Overall sd
## Dim.2 -4.888351        -1.185336 6.542924e-17         0.6542   1.304195
##            p.value
## Dim.2 1.016843e-06
##
## $`3`
##         v.test Mean in category Overall mean sd in category Overall sd
## Dim.1 4.724950        1.5867966 1.779977e-17      0.9262864   1.567094
## Dim.2 1.964313        0.5490124 6.542924e-17      0.9944514   1.304195
##            p.value
## Dim.1 2.301724e-06
## Dim.2 4.949380e-02
```

The above output indicates that: * individuals in cluster 1 have high coordinates on axes 1 and negative ones on axes 2 * individuals in cluster 2 have negative coordinates on axes 2 * individuals in cluster 3 have positive coordinates on axes 1 and 2

Representative individuals of each cluster can be extracted as follows:

```
res.hcpc$desc.ind$para
```

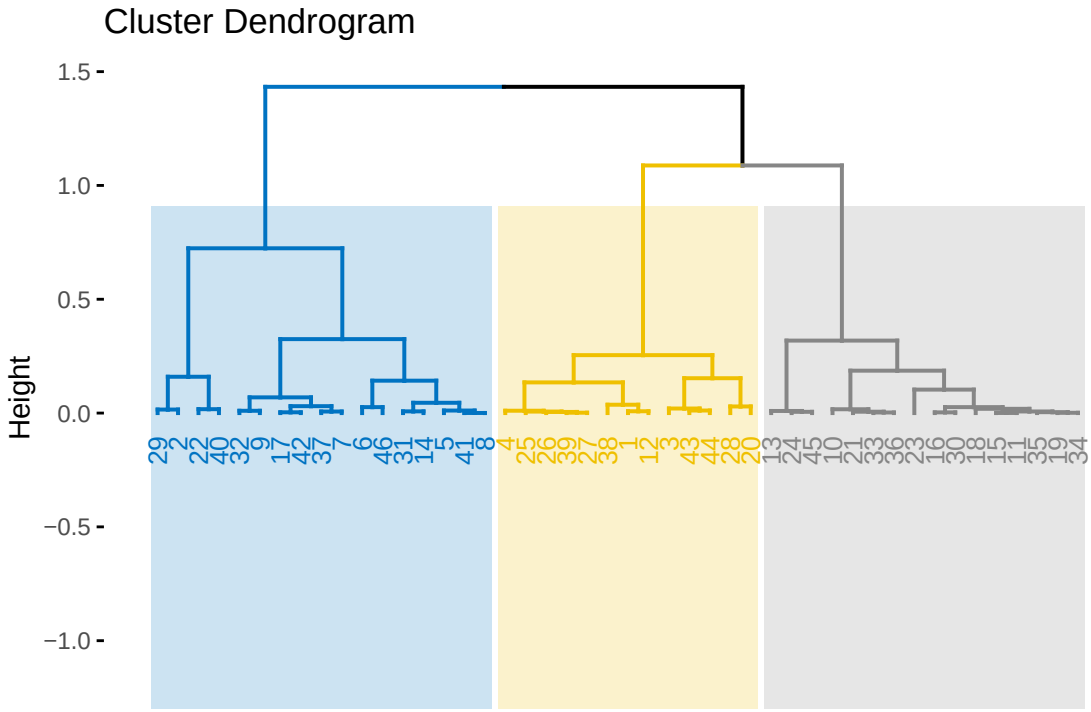
```
## Cluster: 1
##         8        41         5        31        32
## 0.8853125 0.9860347 1.3594052 1.5632877 1.7569712
## ------------------------------------------------------------
## Cluster: 2
##        15        19        16        11        34
## 0.5373540 0.5772885 0.6232877 0.6554539 0.7002065
## ------------------------------------------------------------
## Cluster: 3
##        26        25         3         1        27
## 0.7539972 0.8697866 0.8893865 1.1456330 1.1662802
```

For each cluster, the top 5 closest individuals to the cluster center is shown. The distance between each individual and the cluster center is provided. For example, representative individuals for cluster 1 include individuals number 41, 8, 5, 32, 31

To visualize the dendrogram generated by the hierarchical clustering, we use the function *fviz_dend* of the *factoextra* package:

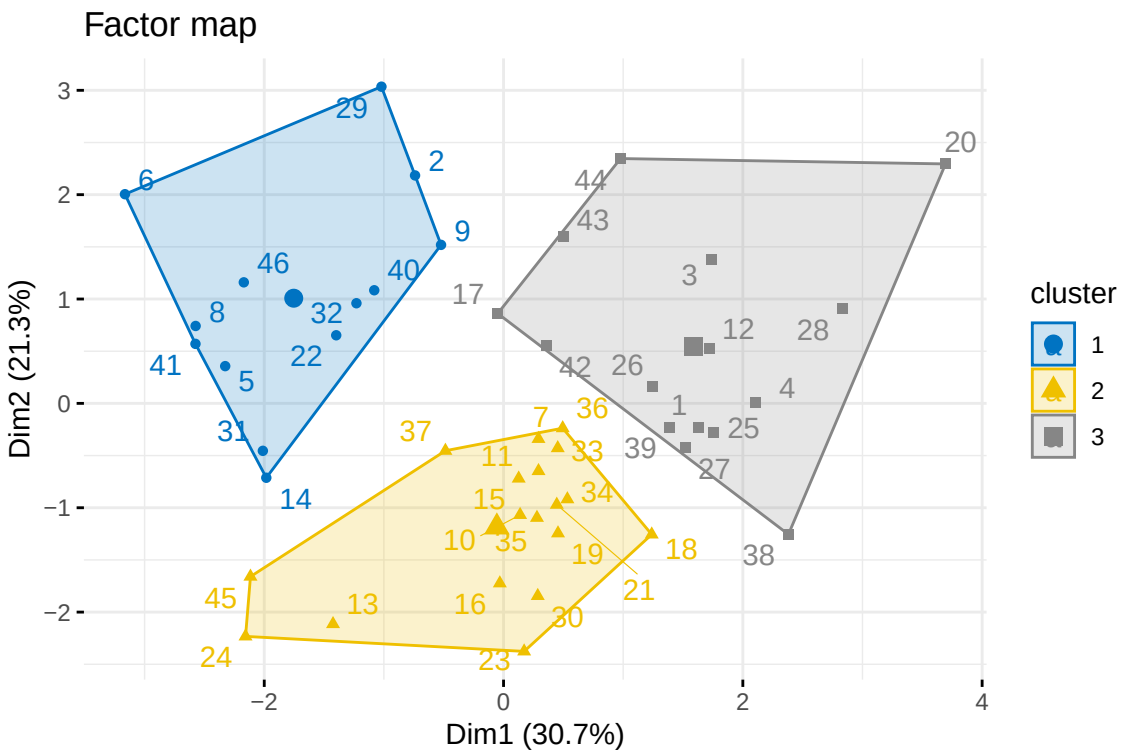
```
fviz_dend(res.hcpc,
          cex = 0.7,             # Label size
          palette = "jco",       # Color palette
          rect = TRUE,
          rect_fill = TRUE, # Add rectangle around groups
          rect_border = "jco",           # Rectangle color
          labels_track_height = 0.8   # Increase the room for labels
          )
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



As we did above, it is possible to visualize individuals on the principal component map and to color individuals according to the cluster they belong to. The function *fviz_cluster* (package *factoextra*) can be used to visualize individuals clusters:

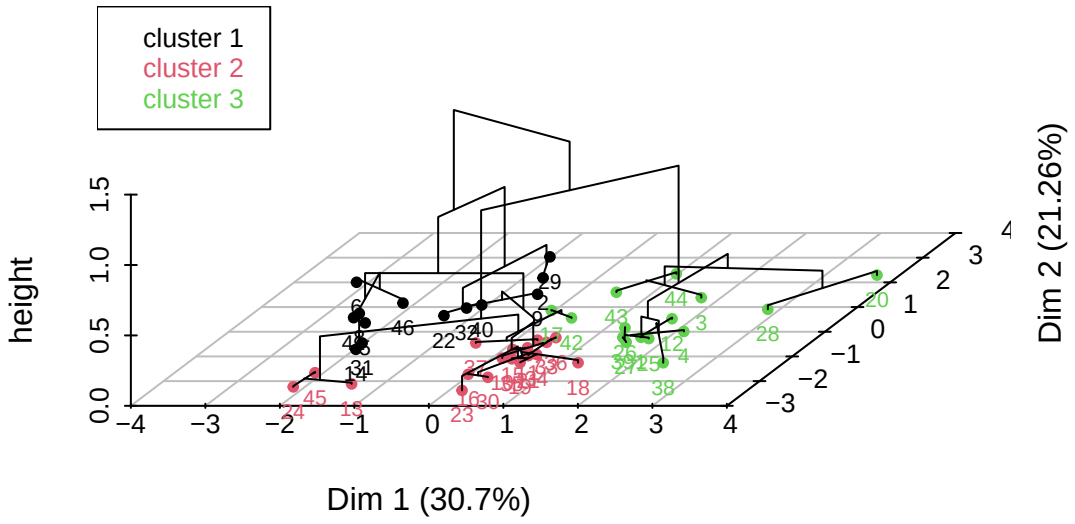
```
fviz_cluster(res.hcpc,
             repel = TRUE,            # Avoid label overlapping
             show.clust.cent = TRUE, # Show cluster centers
             palette = "jco",      # Color palette see
             ggtheme = theme_minimal(),
             main = "Factor map"
             )
```



What we can see from the above plot is that the HCPC method performs a quite good separation of the individuals on 3 clusters, which is perfectly described by the first factorial plane.

We can also combine PCA and tree and depict a 3D plot combining the hierarchical clustering and the factorial map using the R base function *plot*:

```
plot(res.hcpc, choice = "3D.map")
```



3 Partitioning clustering: k-means

Let's now focus on the most famous partitioning clustering algorithm: k-means. It's a stable algorithm based on moving averages where, at each steps, elements are gathered into the closest group (in terms of Euclidean distance for example). Because the algorithm is based on arbitrary starting centroids, we will run it multiple times in order to make the most generalist conclusions possible about our data. To compute this algorithm, we will use the function *kmeans* from the R package *stats*.

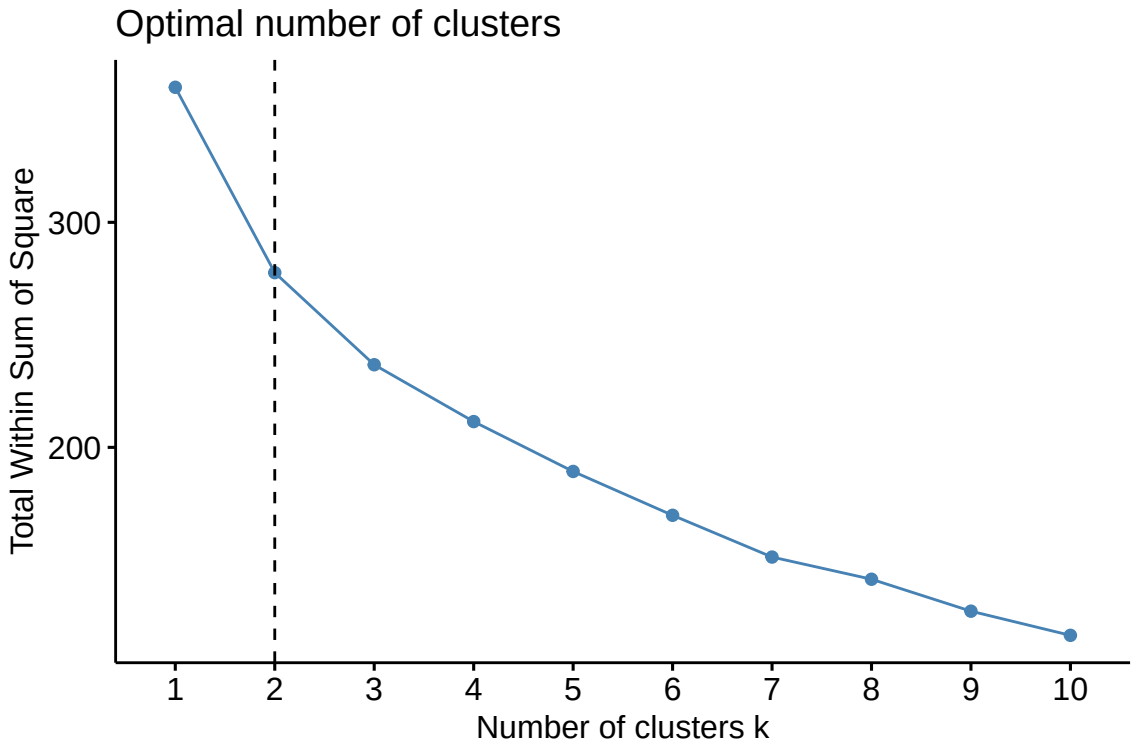
3.1 Finding the number of clusters

In order to have the most relevant method, we first need to find the optimal number of clusters (which is the number of centroids we will start with). We will use various methods: * Elbow method: based on plotting the within-cluster sum of squares (wss) and then identifying the optimal number of clusters (or centers) at the bend (called elbow) of the plot. * Silhouette method: based on how well each object lies within a cluster. * Gap statistic method: based on a null hypothesis (statistical approach), compares the total within-cluster variation using different k values on various samples of the data, using bootstrapping.

```
# set seed for repeating experiments
set.seed(67600)
```

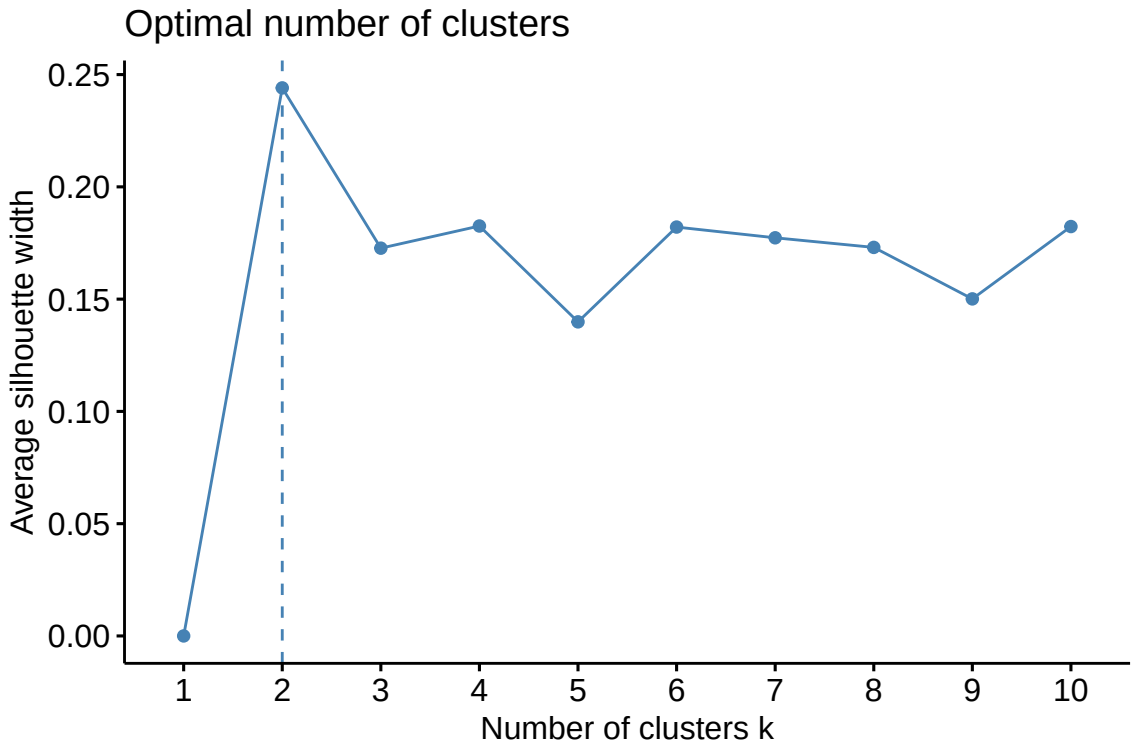
Elbow method:

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



Average silhouette:

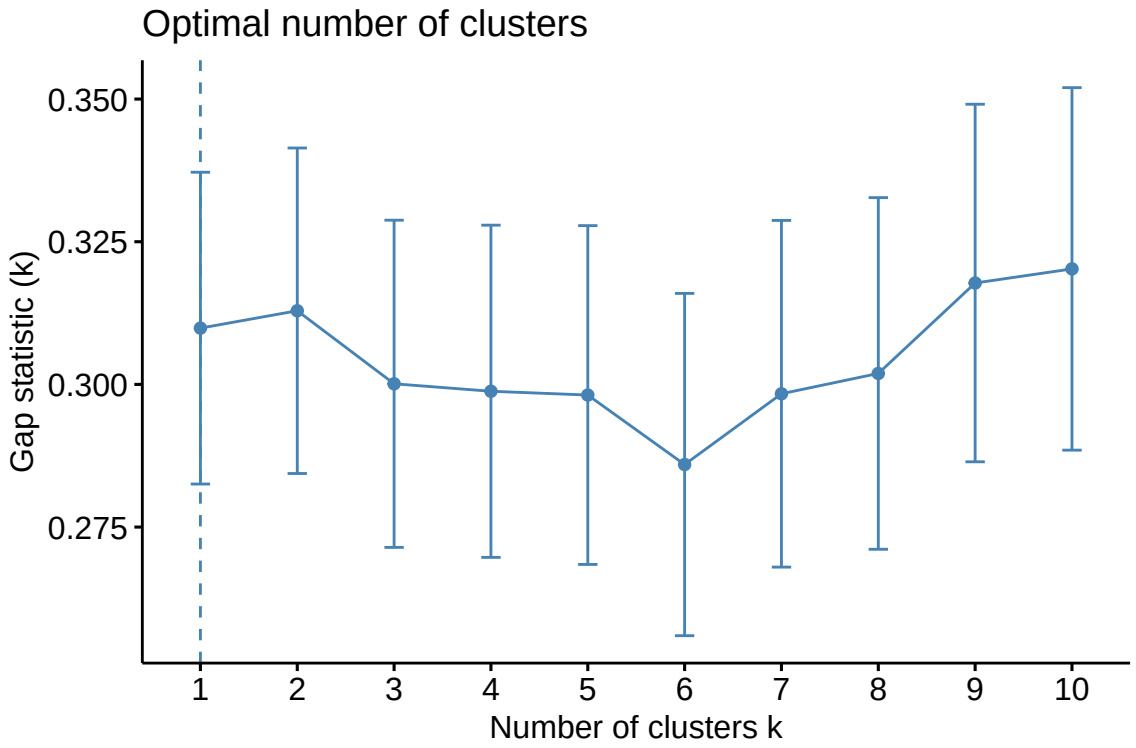
```
fviz_nbclust(Prostate_no_outliers_sc, kmeans, method = "silhouette")
```



Gap statistics:

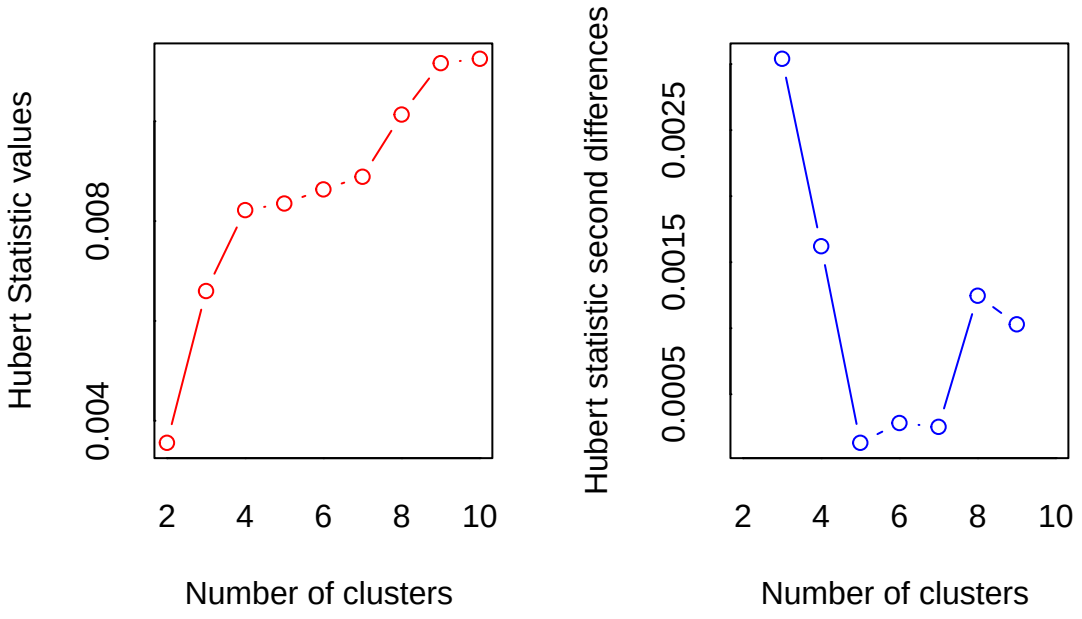
```
# we used B = ~500
gap_stat <- clusGap(Prostate_no_outliers_sc, FUN = kmeans,
```

```
                    nstart = 25, K.max = 10, B = 500)
fviz_gap_stat(gap_stat)
```

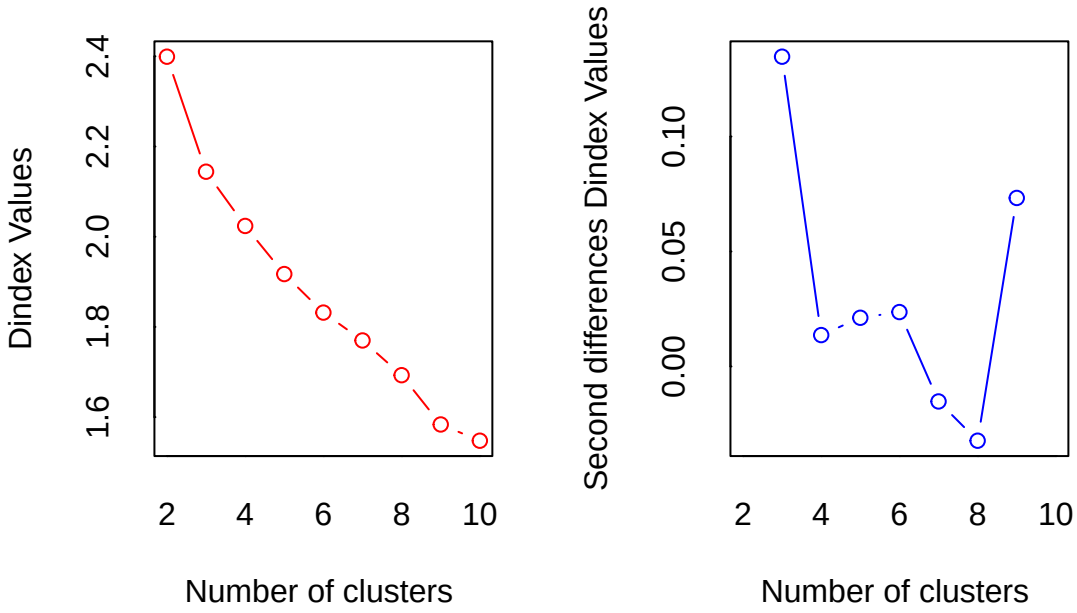


A more robust way could be to use the R package and function *NBclust*, which provides 30 indexes for determining the number of clusters and proposes to use the best clustering scheme from the different results obtained by varying all combinations of number of clusters, distance measures, and clustering methods:

```
library(NbClust)
nc <- NbClust(Prostate_no_outliers_sc, 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:
## * 1 proposed 2 as the best number of clusters
## * 11 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 3 proposed 5 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 3 proposed 9 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  3
##
##
## *******************************************************************
```

We'll show the clusters for k=2 and k=3, which is the optimal value suggested by *NBclust*.

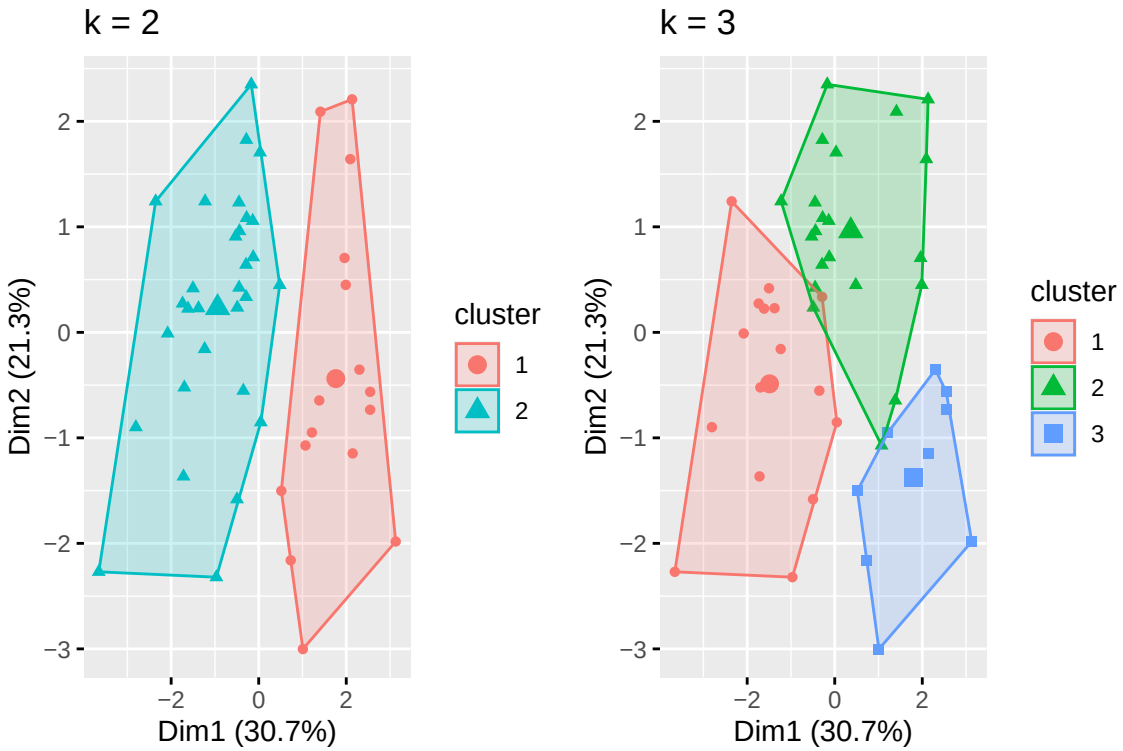
```
k2 <- kmeans(Prostate_no_outliers_sc, centers = 2, nstart = 25)
k3 <- kmeans(Prostate_no_outliers_sc, centers = 3, nstart = 25)
```

We create the graphs side-by-side:

Table 6: Mean by group and by variable

cluster	age	wt	sbp	dbp	hg	sz	sg	ap
1	0.2160218	0.5206500	0.7683977	1.0181135	0.6959199	-0.1920443	-0.2422403	-0.3709806
2	-0.1587314	-0.2108180	-0.5921312	-0.6843484	-0.3031253	-0.3080829	-0.2703321	-0.4717765
3	-0.0136656	-0.4336914	0.0155992	-0.2131668	-0.5298985	1.0602723	1.0614243	1.7603329

```
p1 <- fviz_cluster(k2, geom = "point", data = Prostate_no_outliers_sc) + ggtitle("k = 2")
p2 <- fviz_cluster(k3, geom = "point",  data = Prostate_no_outliers_sc) + ggtitle("k = 3")

grid.arrange(p1, p2, nrow = 1)
```



From the plot above, we can graphically visualize the 2 explored cases and it is clearly visible that the case with k=2 better separates the observations, without any overlapping. Since most of the methods suggested the case with k=3 (which is also the value we obtained in the hierarchical algorithms), let's do some more in-depth analysis to understand the meaning of the obtained clusters.

We can compute the mean of each variables by cluster using the original data:

```
aggregate(Prostate_no_outliers_sc, by=list(cluster=k3$cluster), mean) %>%
    kbl(caption = "Mean by group and by variable") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

From the above table we can see the following: * Cluster 1 contains observations with (on average) negative values along all the variables. * Cluster 2 has positive averages for each variable, except for the last 3. * Cluster 3 contains elements characterized by high values of the variables 'sz', 'sg' and 'ap'.

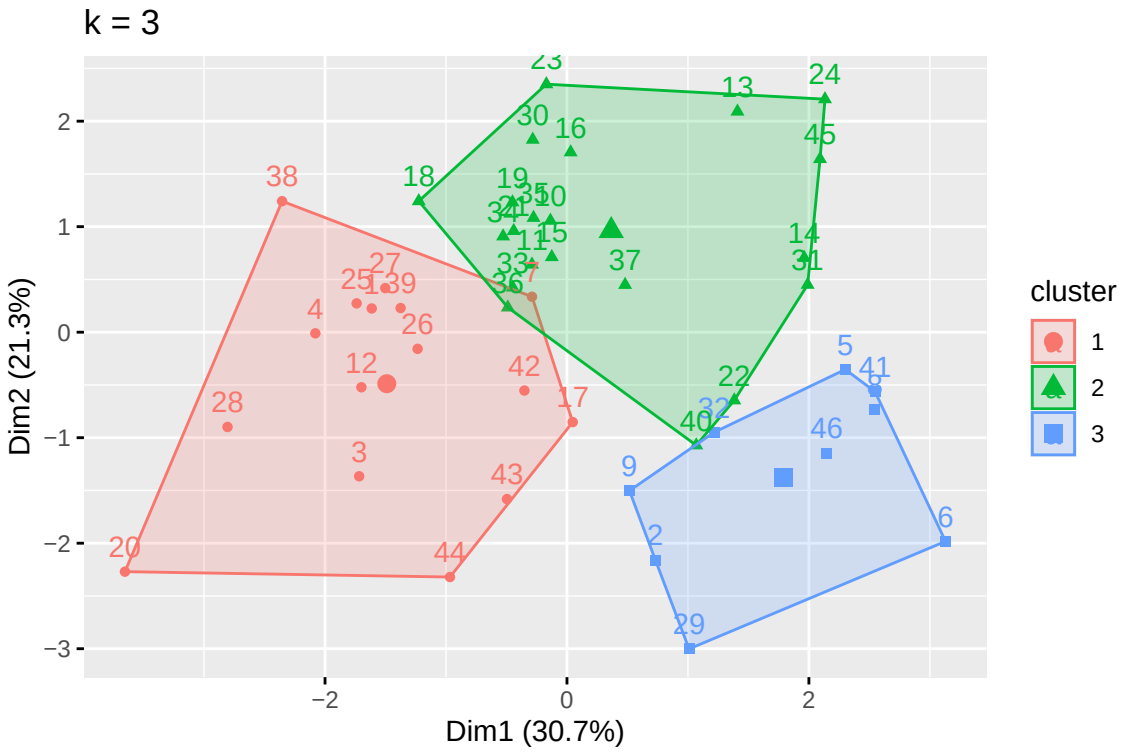
Let's check the cluster size:

```
k3$size
```

```
## [1] 16 21  9
```

and the graph:

```
fviz_cluster(k3,  data = Prostate_no_outliers_sc) + ggtitle("k = 3")
```



4 Hierarchical K-Means Clustering

K-means, as we know, is very sensitive to the initial choices made regarding the number of clusters we want to find and to the initial positioning of the centroids. Therefore, one possible improvement that can be made is to integrate a hierarchical clustering approach in the initial stage of the k-means algorithm, to better initialize the centroids when applying k-means. This is what the hierarchical k-means clustering algorithm does, allowing to obtain more robust clusters with respect to the classical k-means.

Let's apply the algorithm:

```
res.hk <-hkmeans(Prostate_no_outliers_sc,3)
```

Print the results:

```
res.hk
```

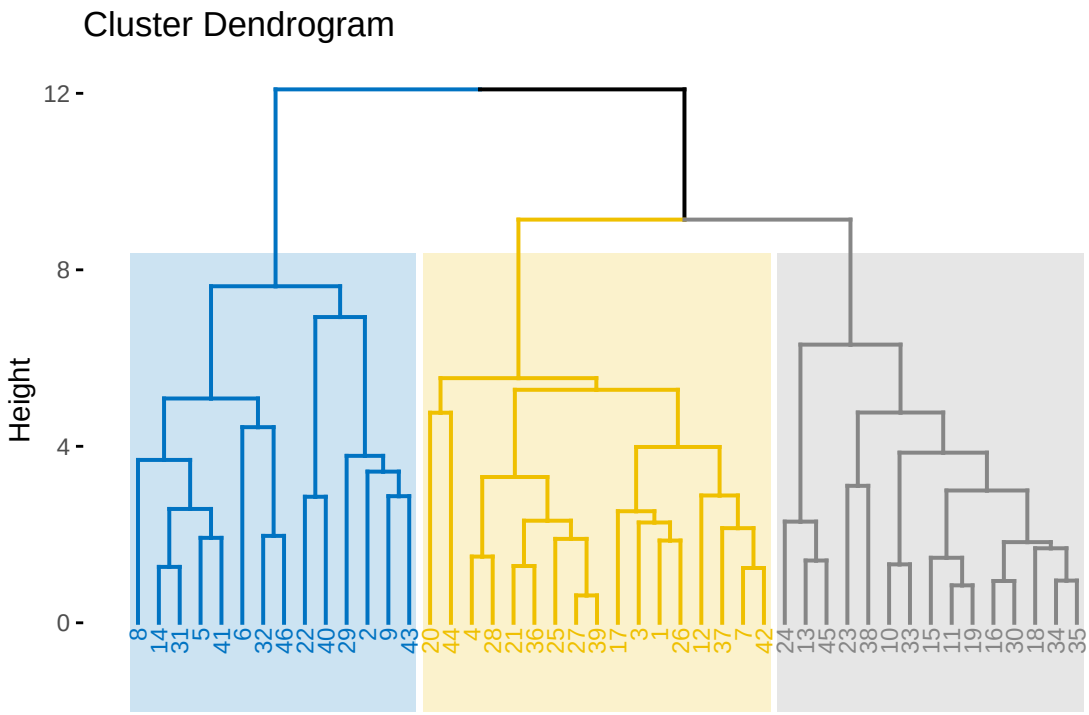
```
## Hierarchical K-means clustering with 3 clusters of sizes 17, 13, 16
##
## Cluster means:
##          age          wt         sbp         dbp          hg          sz          sg
## 1  0.1882887  0.50642363  0.7090406  0.9561726  0.6905556 -0.2369977 -0.1998273
## 2 -0.4897788 -0.61417705 -0.1342213 -0.5285940 -0.3450701  0.7178314  1.0838337
## 3  0.1978886 -0.03905626 -0.6443008 -0.5864508 -0.4533458 -0.3314280 -0.6682984
##           ap
## 1 -0.3813673
## 2  1.1483270
## 3 -0.5278129
##
```

```
## Clustering vector:
##  [1] 1 2 1 1 2 2 1 2 2 3 3 1 3 2 3 3 1 3 3 1 3 2 3 3 1 1 1 1 2 3 2 2 3 3 3 1 3 1
## [39] 1 2 2 1 1 1 3 2
##
## Within cluster sum of squares by cluster:
## [1]  75.08889 105.01228  56.64030
##  (between_SS / total_SS =  34.2 %)
##
## Available components:
##
##  [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
##  [6] "betweenss"    "size"         "iter"         "ifault"       "data"
## [11] "hclust"
```

Visualize the dendrogram:

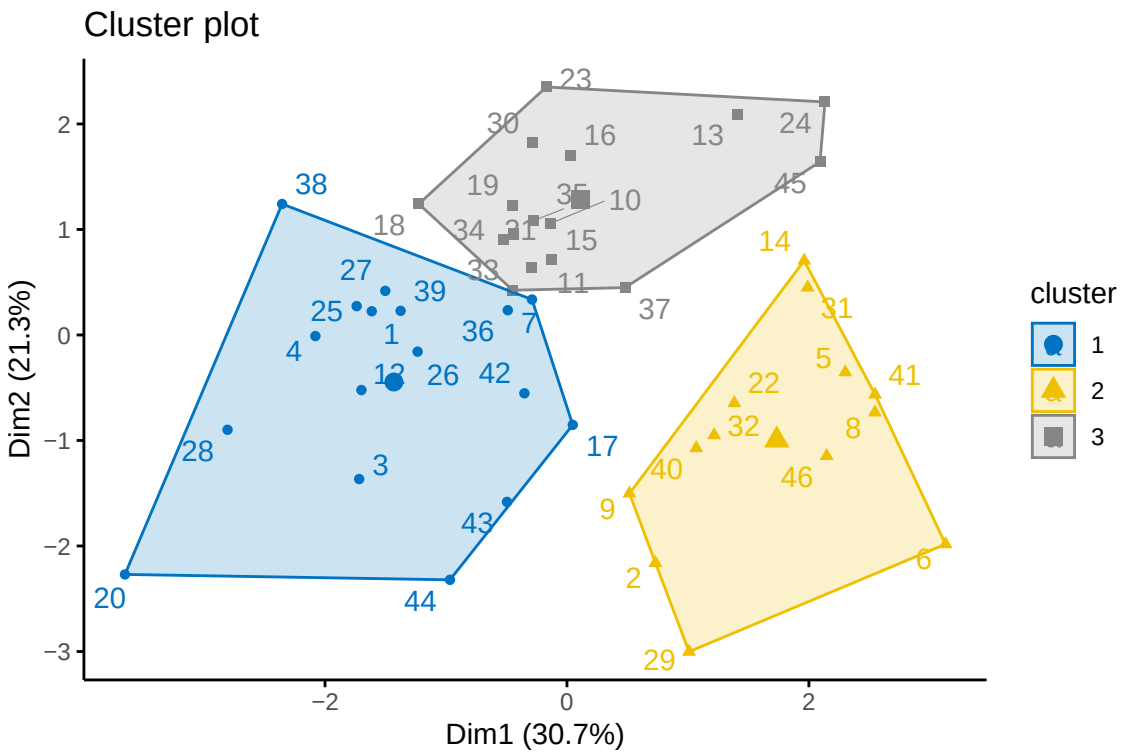
```
fviz_dend(res.hk, cex = 0.6, palette = "jco",
          rect = TRUE, rect_border = "jco", rect_fill = TRUE)
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



Visualize the final clusters:

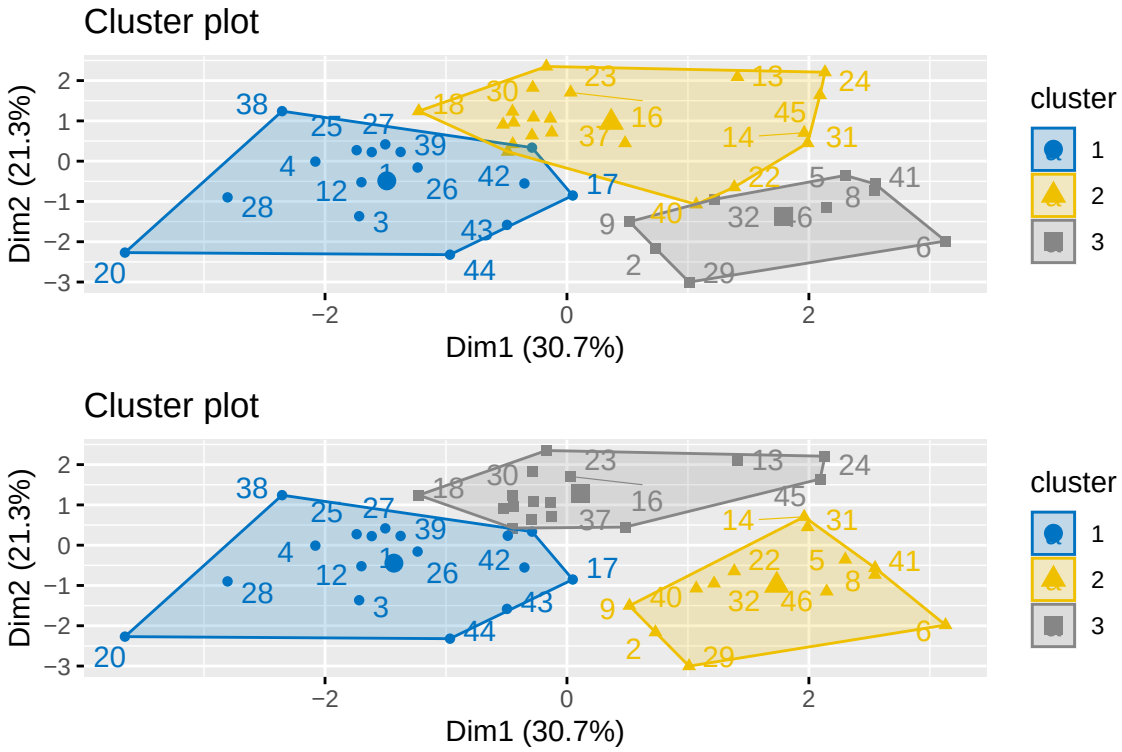
```
fviz_cluster(res.hk, palette = "jco", repel = TRUE,
             ggtheme = theme_classic())
```



Let's visualize one beside the other the results of k-means and the hierarchical k-means we just performed, to visualize the eventual differences:

```
g1 <- fviz_cluster(k3,  data = Prostate_no_outliers_sc, palette = "jco", repel = TRUE)
g2 <- fviz_cluster(res.hk, palette = "jco", repel = TRUE)
grid.arrange(g1, g2)
```

```
## Warning: ggrepel: 10 unlabeled data points (too many overlaps). Consider increasing max.overlaps
## ggrepel: 10 unlabeled data points (too many overlaps). Consider increasing max.overlaps
```



The plot shows that the introduction of the hierachical clustering initialization in the k-means improves the separation between the clusters already found with classical k-means. Indeed the 3 groups are more separated in the 2nd case.

5 Model-based clustering

In model-based clustering, the data is considered as coming from a mixture of distributions.

In our case, we consider a Gaussian mixture model, where each component k in the model can be considered as a cluster and is modeled by a normal distribution

Model-based clustering doesn't require to do a standardization, but we'll use the standardized data to be able to compare the results of the different algorithms.

We'll use the R package *mclust* which estimates the model's parameters by EM algorithm initialized by hierarchical model-based agglomerative clustering. The optimal model is then selected according to the BIC criteria:

The function we'll use describes the obtained model with a character string composed by 3 characters indicating the characteristics of the model at which the optimal BIC occurs: * the first character refers to volume * the second to shape * the third to orientation

In details: character E stands for "equal", V for "variable" and I for "coordinate axes".

Let's now load the library of interest:

```
library(mclust)
```

We can now perform the fitting and print a summary of the obtained model:

```
mc <- Mclust(Prostate_no_outliers_sc)
summary(mc) # Print a summary
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVI (diagonal, varying volume and shape) model with 2 components:
##
##  log-likelihood  n df       BIC       ICL
##       -378.9979 46 33 -884.3409 -884.4017
##
## Clustering table:
##  1  2
## 28 18
```

The lowest the BIC, the better the model fits to the data: in this case we have a strongly negative BIC. The above output shows that the optimal model returned by the algorithm is composed by 2 clusters with respectively 28 and 18 observations. We therefore obtained a different result with respect to the previously considered methods which return 3 clusters in most of the cases. We think this is a more realistic scenario, considering that our data is about patients that have either stage 3 or stage 4 prostate cancer: therefore we suppose that the found clusters separate people with this two cancer stages. The optimal selected model name is a VVI model: it means that the 2 spotted clusters are ellipsoidal with varying volume, varying shape and the orientation is the identity (I) or "coordinate axes".

To better understand the results of the algorithm, we can analyze one by one the outputs of the *Mclust* function:

Name of the optimal model:

```
mc$modelName
```

```
## [1] "VVI"
```

Number of clusters in the optimal model:

Table 7: Probability to belong to a given cluster (sample of 10)

V1	V2
0.9985860	0.0014140
0.0000000	1.0000000
0.9985823	0.0014177
1.0000000	0.0000000
0.9999569	0.0000431
0.9996022	0.0003978
0.9999941	0.0000059
0.0000000	1.0000000
0.9999960	0.0000040
0.9999919	0.0000081

```
mc$G
```

```
## [1] 2
```

Probability to belong to a given cluster:

```
mc$z  %>%
    as.data.frame(.) %>%
  sample_n(., 10, replace=FALSE) %>%
  kbl(caption = "Probability to belong to a given cluster (sample of 10)") %>%
  kable_classic(full_width = F, html_font = "Cambria")
```

We know that model-based clustering is a soft-clustering technique: therefore we have for each sample a probability to belong to each one of the clusters. The above table, shows that for this sample of data, the assignment of points to clusters is pretty strong, since we have high probabilities for each observation to belong to a single specific cluster.

We can also visualize the cluster allocation of each observation:

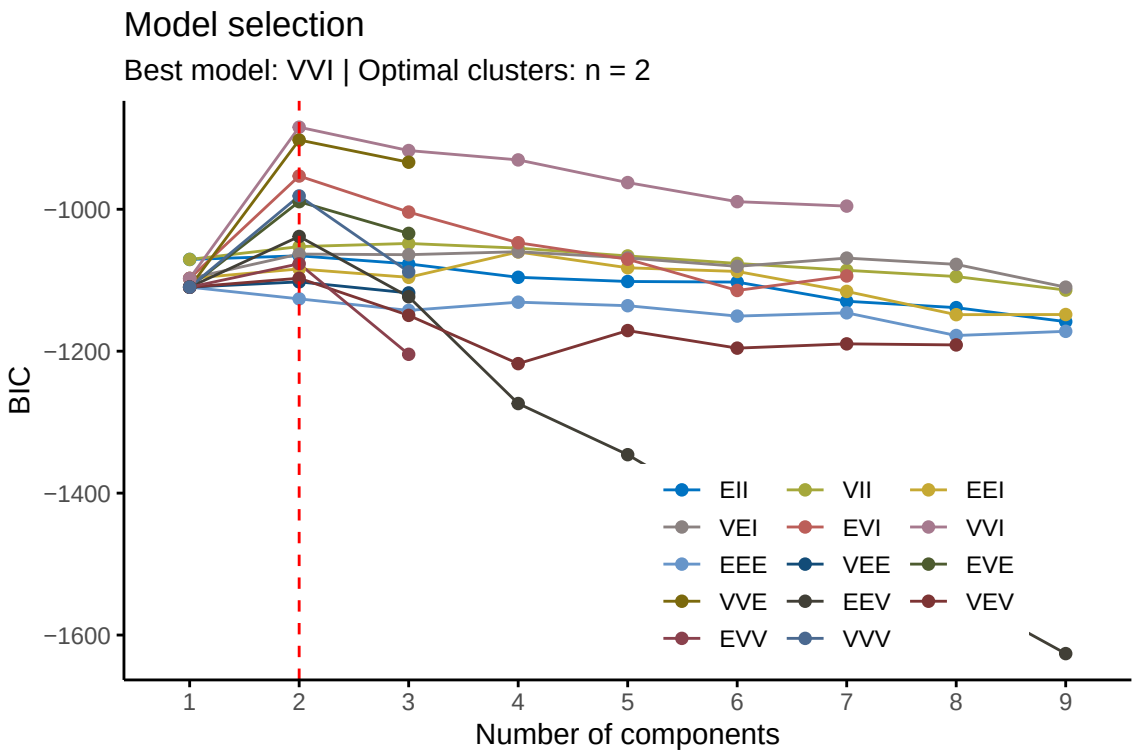
```
head(mc$classification,10) # first 10 observations
```

```
##  [1] 1 2 1 1 2 2 1 2 2 1
```

We can use the function *plot.Mclust* in the *mclust* package to plot the results of the algorithm. Some graphs can be plotted by using the *factoextra* package, which uses *ggplot2* providing a better plotting style.

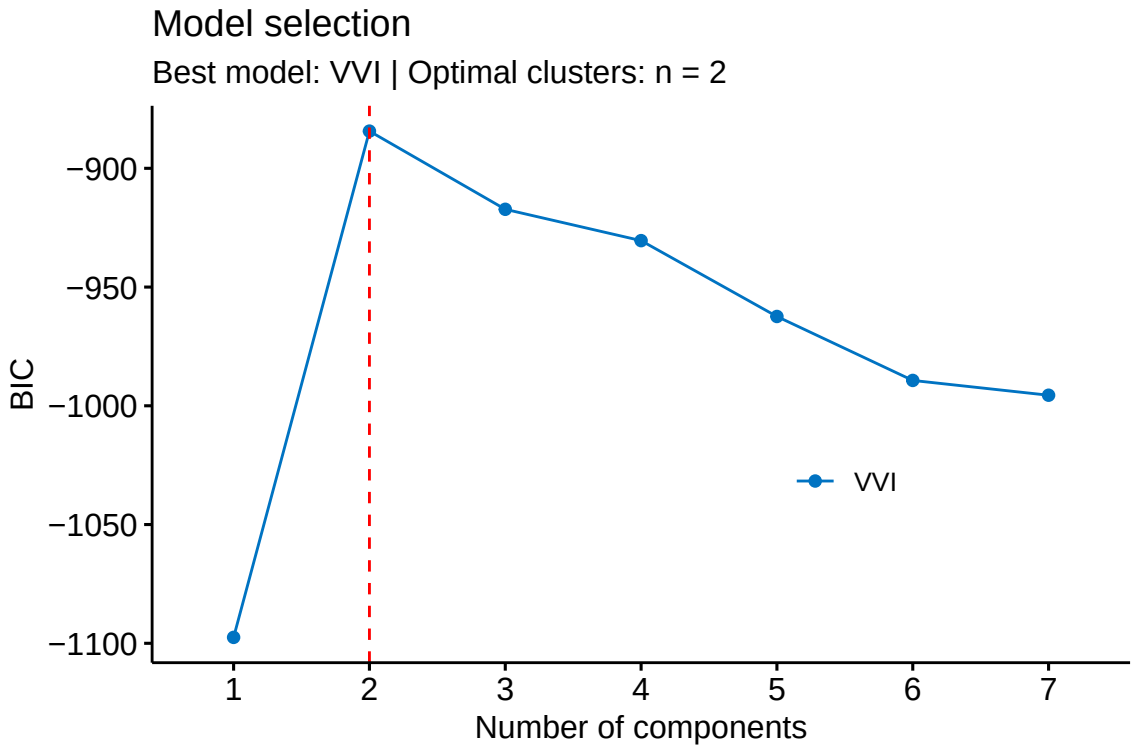
We can graph the *BIC values* used for choosing the number of clusters:

```
library(factoextra)
fviz_mclust(mc, "BIC", palette = "jco")
```



We can modify the above graph to show only the final optimal model we are interested in using the function *fviz_mclust_bic*:

```
library(factoextra)
fviz_mclust_bic(mc, model.names=mc$modelName, palette = "jco")
```



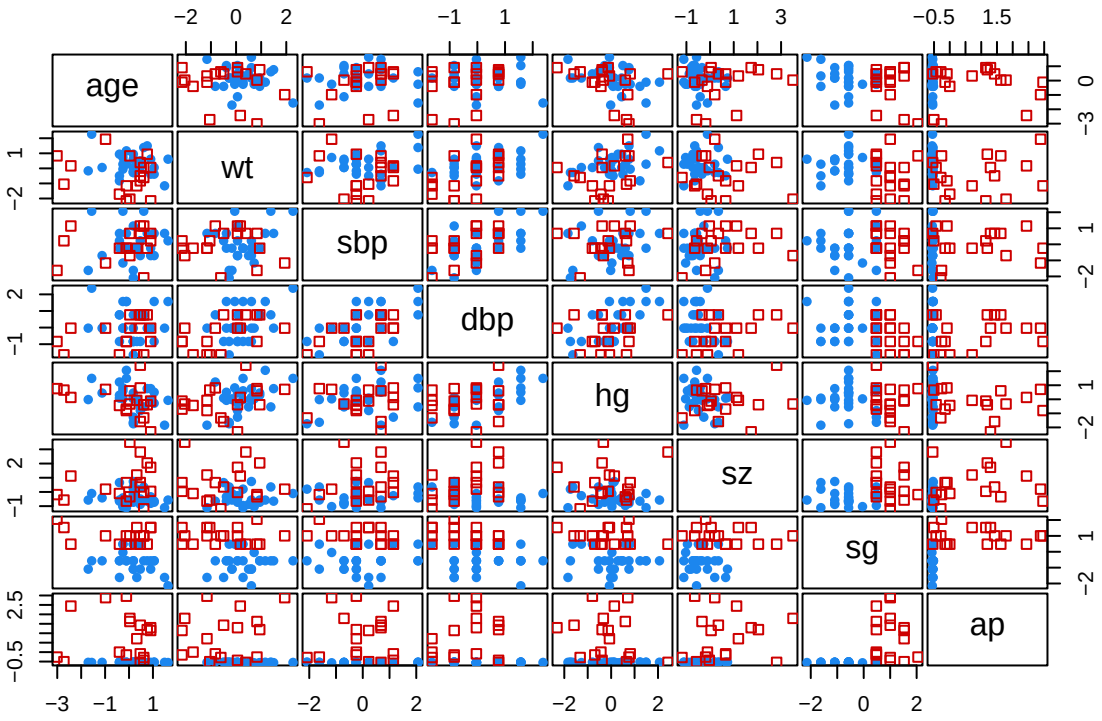
From *mclust*, we can obtain the top BIC ranking:

```
summary(mc$BIC)
```

```
## Best BIC values:
##              VVI,2      VVE,2      VVI,3
## BIC      -884.3409 -902.43344 -917.26093
## BIC diff    0.0000  -18.09254  -32.92002
```

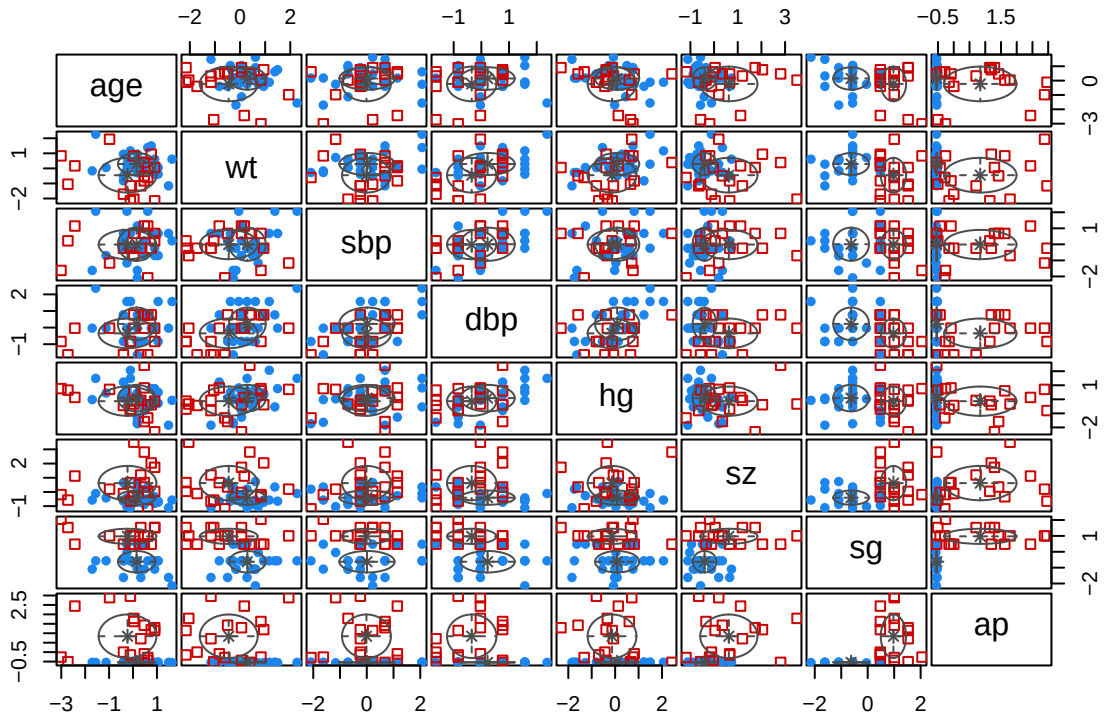
By specifying the 'classification' paramerer we chan visualize a plot showing the clustering.

```
plot.Mclust(mc, what="classification", addEllipses = FALSE)
```



Ellipses corresponding to covariances of mixture components are also drawn if *addEllipses = TRUE.*:

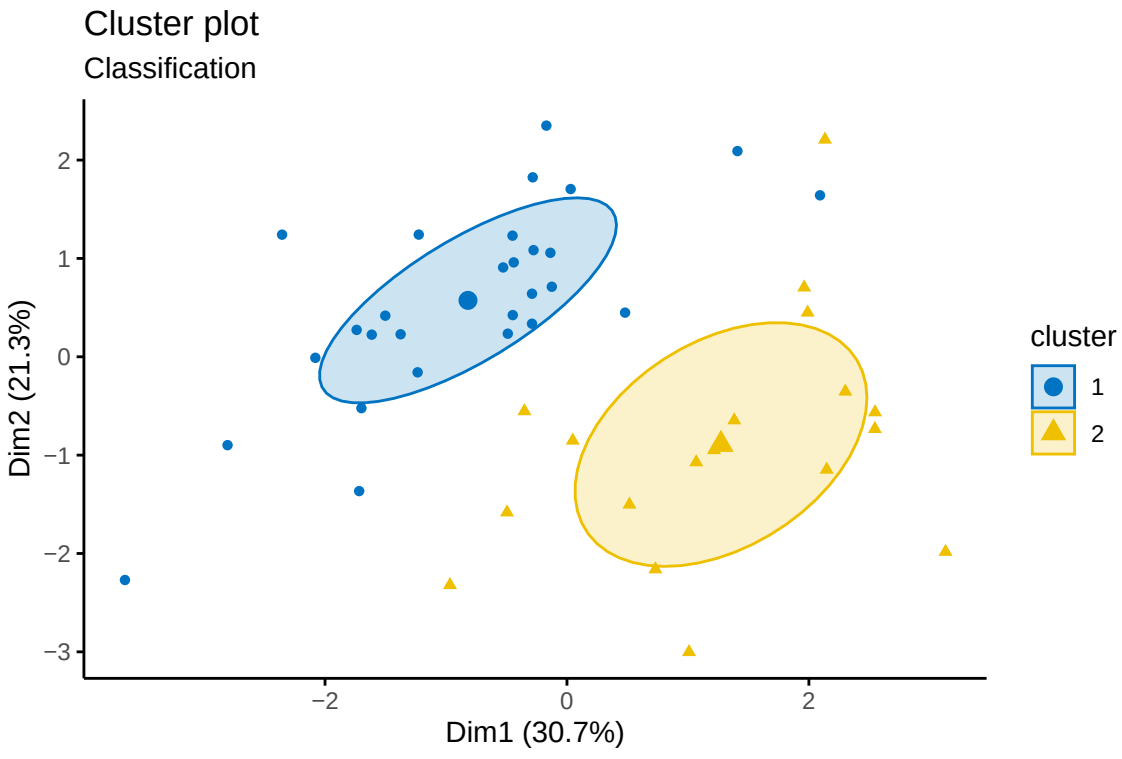
```
plot.Mclust(mc, what="classification", addEllipses = TRUE)
```



The above plots are not very clear due to the high number of variables used for performing the clustering: indeed they show the found clusters in the graph with each group of 2 variables. The above plots show for instance that variables 'sg' and 'ap' are very discriminatory for the separation of points into the 2 groups.

We may prefer to show the clustering result in the first factorial plane (obtained through PCA). To do that we can use the *factoextra* package wich performs a PCA to show the results in the first 2 PCs:

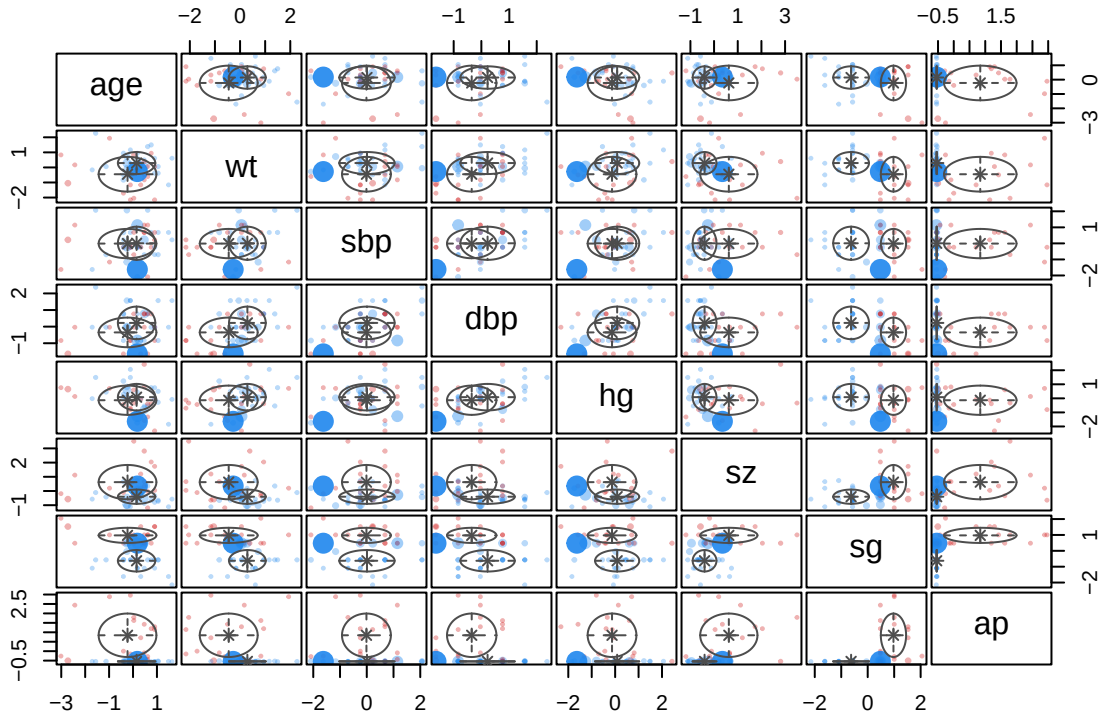
```
fviz_mclust(mc, "classification", geom = "point",
            pointsize = 1.5, palette = "jco")
```



In the 1st factorial plane the 2 clusters are pretty well separated by the diagonal of the 1st quadrant.

We can now plot the classification uncertainty:

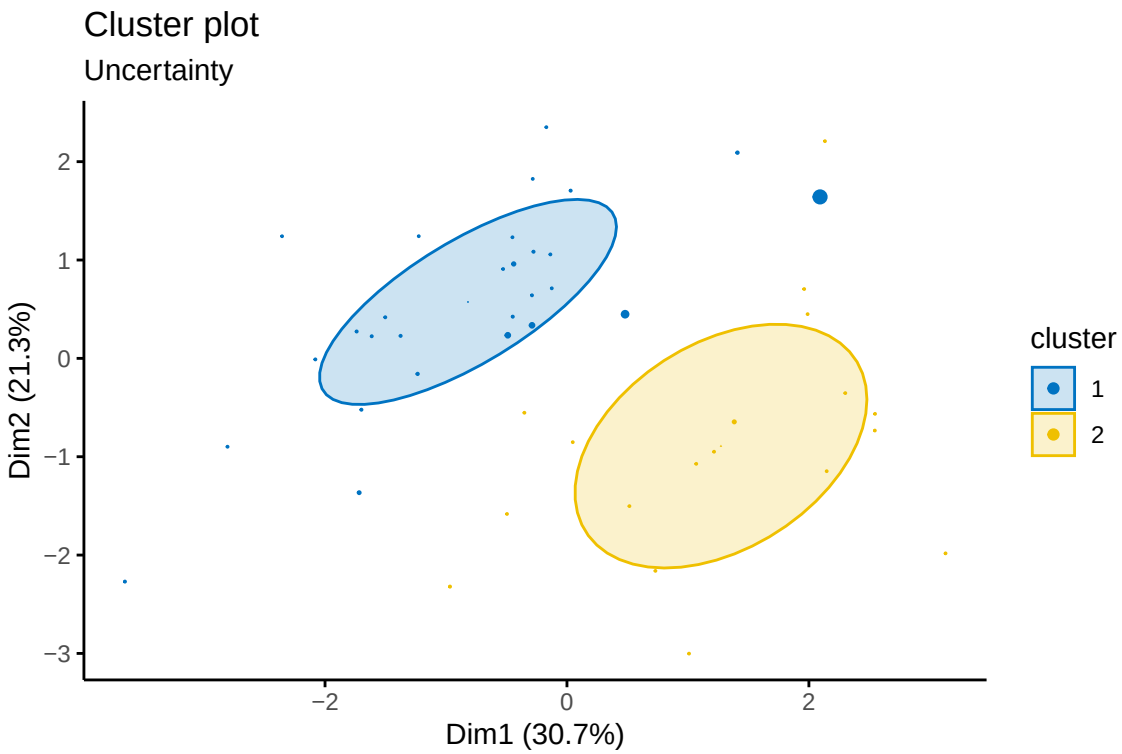
```
plot.Mclust(mc, what="uncertainty")
```



In this plot, *larger symbols indicate the more uncertain observations*. The package *factoextra* provides a better visualization for classification uncertainty on the first factorial plane (first 2 PCs):

```
fviz_mclust(mc, "uncertainty", palette = "jco")
```

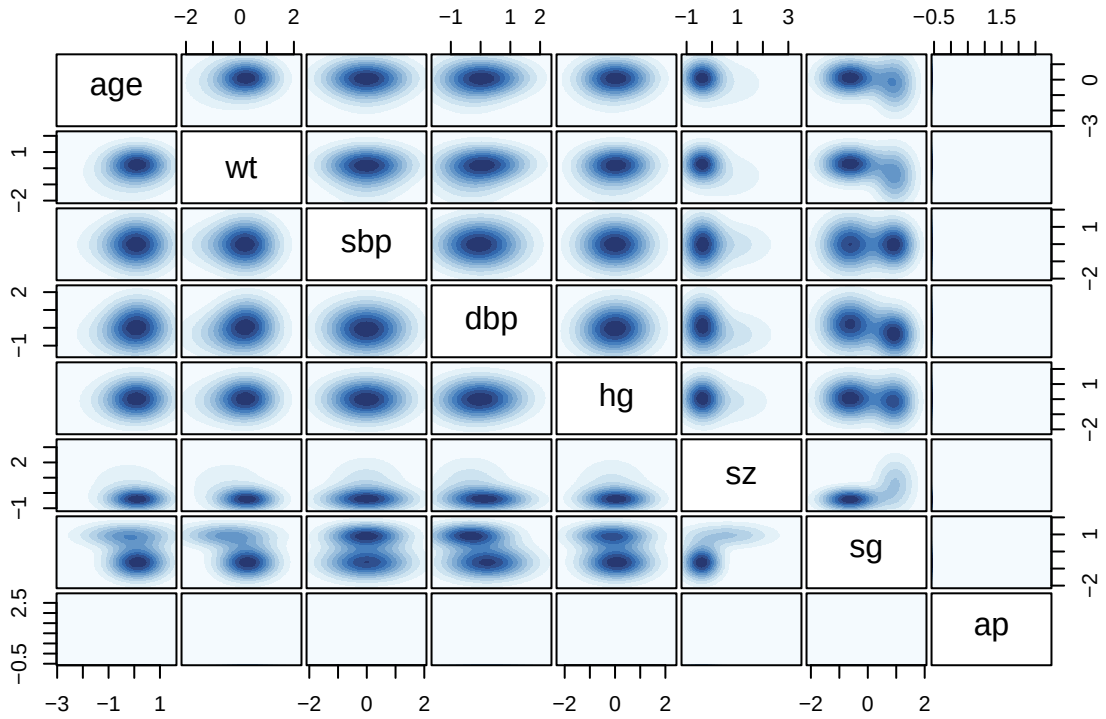
```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



The plot shows that there are some points outside the 2 represented densities, with one of them (around position (2,2)) showed with high size, indicating a pretty strong uncertainty in its cluster association.

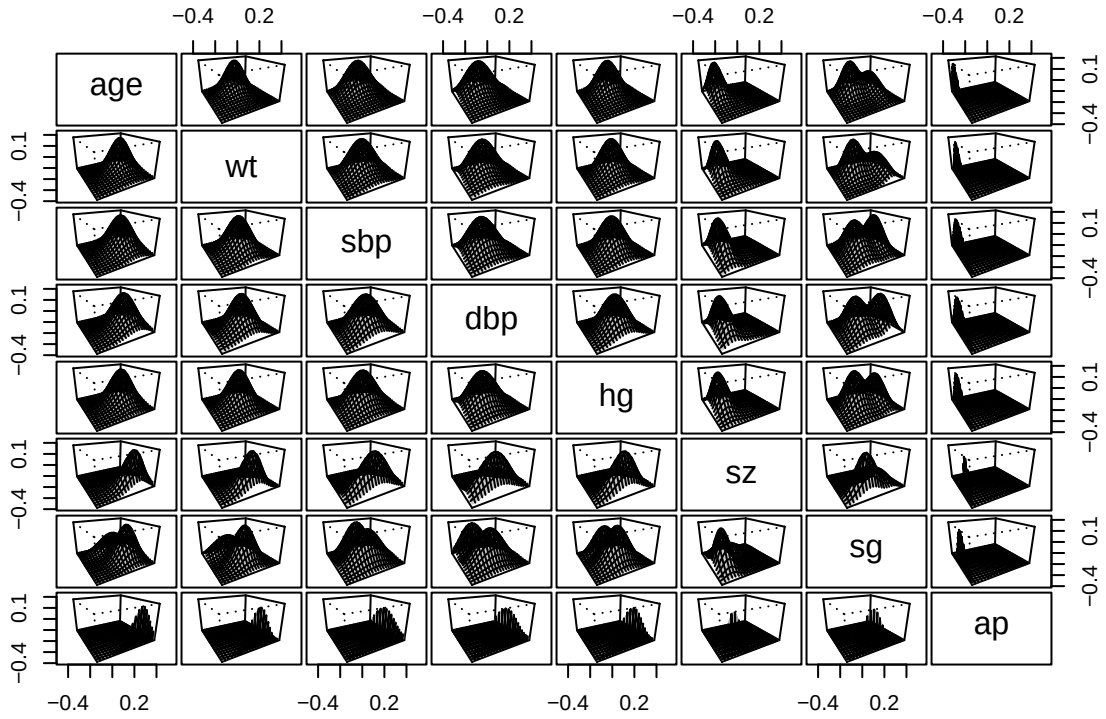
We can also visualize a plot of the estimated density:

```
plot.Mclust(mc, what="density", type = "image", col="steelblue", grid = 200)
```



We can also show 3D graphs of the estimated densities:

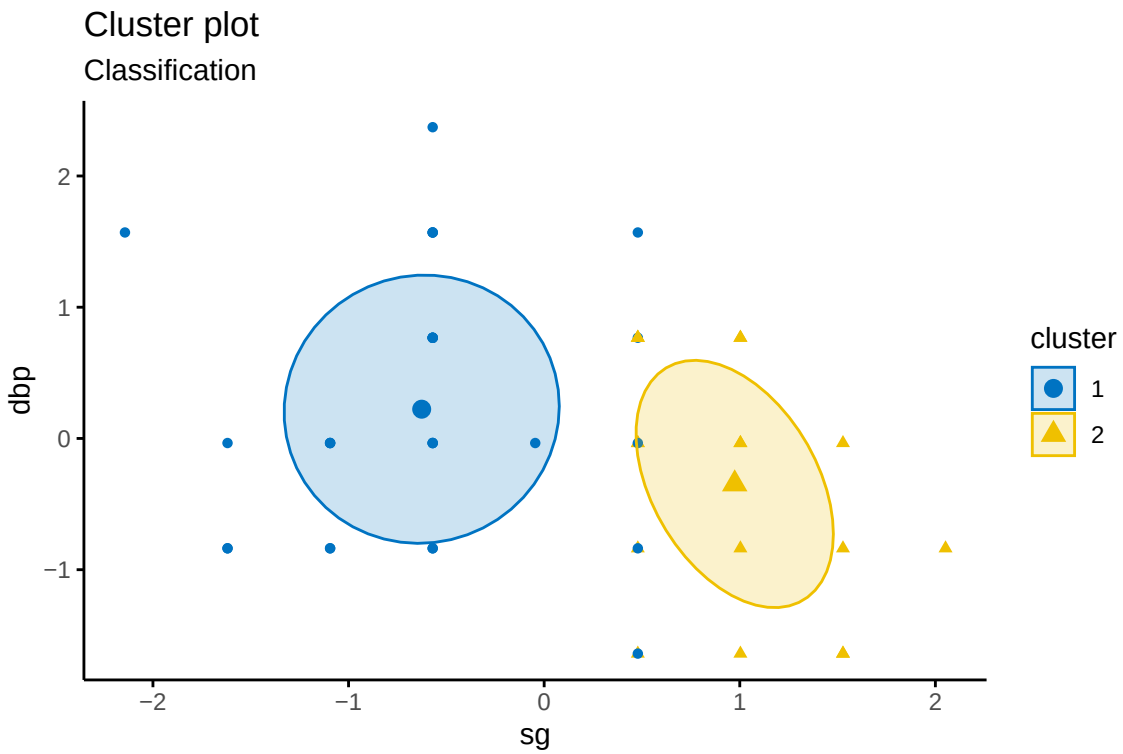
```
plot.Mclust(mc, what="density", type="persp")
```



The density estimation shows the estimated densities by considering each pair of variables: we can see that only some planes show 2 estimated Gaussians, while the planes of some variables are not able to discriminate the 2 clusters. For instance, all the plots considering variable 'sg' show very clearly the 2 densities.

To better show the results we discussed above, we may plot the data using only two variables of interest: we can select 'sg' as one of them (since, as said before, it discriminates very well the 2 groups) and 'dbp':

```
# Classification: plot showing the clustering
fviz_mclust(mc, "classification", geom = "point",
            pointsize = 1.5, palette = "jco",
            choose.vars = c("sg", "dbp"))
```



As expected, these 2 variables are pretty good in discrimating the 2 groups.

It is possible to use also another dimensionality reduction method for visualizing the clustering structure obtained from a finite mixture of Gaussian densities with the function *MclustDR*.

The estimated directions which span the reduced subspace are defined as a set of *linear combinations of the original features, ordered by importance as quantified by the associated eigenvalues*:

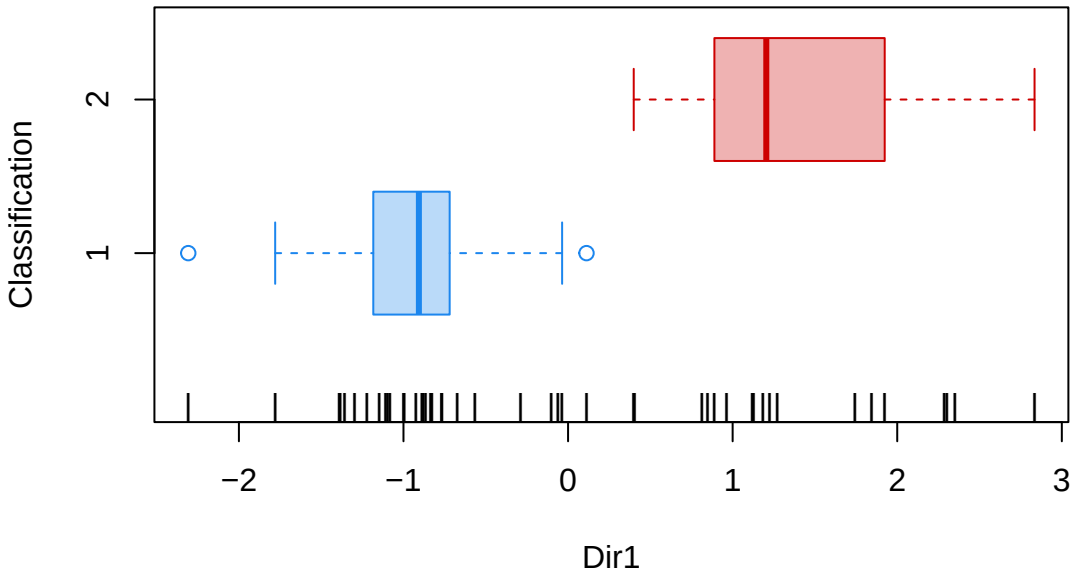
```
drmc <- MclustDR(mc, lambda = 1)
summary(drmc)
```

```
## -----------------------------------------------------------------
## Dimension reduction for model-based clustering and classification
## -----------------------------------------------------------------
##
## Mixture model type: Mclust (VVI, 2)
##
## Clusters  n
##        1 28
##        2 18
##
## Estimated basis vectors:
##          Dir1
## age  0.070479
## wt  -0.196547
## sbp  0.038618
## dbp -0.165069
## hg   0.204623
## sz   0.244341
## sg   0.754339
## ap   0.507014
##
##                Dir1
## Eigenvalues   1.2384
## Cum. %      100.0000
```

In our case, the above method reduced the data to only 1 direction, which captures most of the clustering structure found with the model-based algorithm.

The projected plot on this new direction would be:

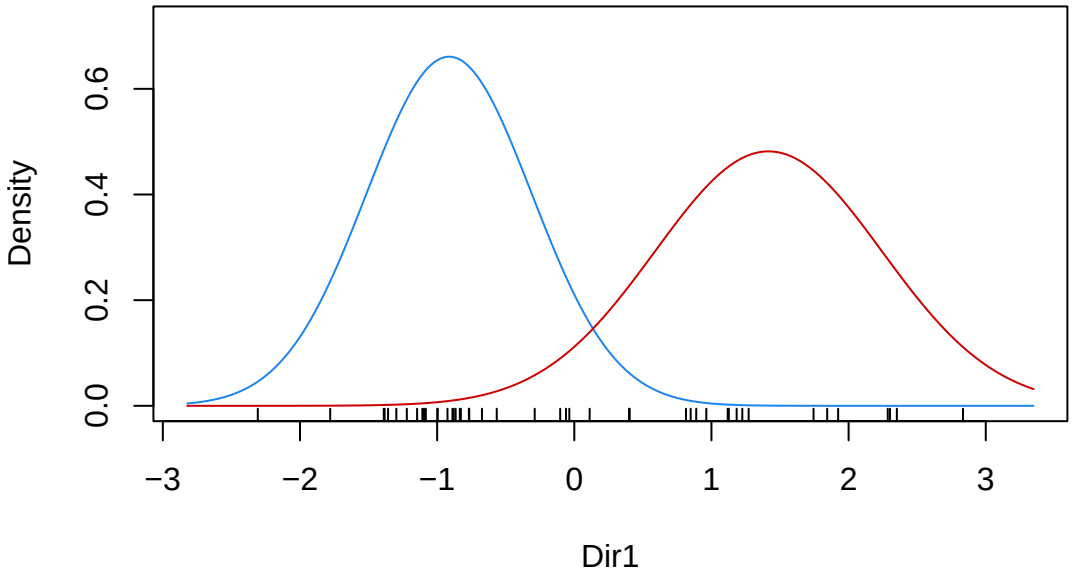
```
plot(drmc, what = "contour")
```



The plot allows to visualize, with a sort of box-plot, also the uncertainty in assigning the observations into the 2 clusters. The box-plots show also the values in the new direction (Dir1) of the observations in each cluster, and we can see that the data are very well classified into 2 groups.

We can also visualize the density:

```
plot(drmc, what = "density")
```



Save image of all objects created in the session:

```
save.image(file="Clustering.RData")
```