UNIVERSITÀ degli STUDI di CATANIA | DIPARTIMENTO di ECONOMIA e IMPRESA

JUNE 14, 2022

Data Analysis Report:

Work on Dataset:

"80 Cereal"

Raza Akbar
ID: 1000039444

SUBMITTED TO:

Prof. Antonio Punzo

Data Description

Consume a healthy eating pattern that accounts for all foods and beverages within an appropriate calorie level. The '80 Cereals' Dataset describing 80 commonly available breakfast cereals, based on the information that is available on the newly mandated F&DA food label. This Dataset is available on https://www.kaggle.com/crawford/80- cereals, but it was slightly modified before applying the different analysis. It contains 77 rows, each row has a different kind of cereal and 11 columns. In this Dataset a lot of Knowledge about how to understand and choose a good breakfast cereal. In the first column it is a categorical variable that describe the name of the cereal and the other 10 columns describe all the nutritional level as follow:

1. Name
2. Calories
3. Protein
4. Fat
5. Sodium
6. Fiber
7. Carbo
8. Sugars
9. Potassium
10. Vitamins
11. Weight

```
PATH="${D:rtools40}usrbin;${PATH}"
write('PATH="${D:rtools40}\\usr\\bin;${PATH}"', file = "~/.Renviron", append
= TRUE)
Sys.which("make")

## make
##   ""

## "D:\\rtools40\\usr\\bin\\make.exe"
```

Exploration of Dataset

Now, we will see the complete structure of the dataset and also make some transformations in the dataset if needed so.

```
str(cereal)
```

```
## 'data.frame':    77 obs. of  11 variables:
##  $ name    : chr  "100% Bran" "100% Natural Bran" "All-Bran" "All-Bran
with Extra Fiber" ...
##  $ calories: int  70 120 70 50 110 110 110 130 90 90 ...
##  $ protein : int  4 3 4 4 2 2 2 3 2 3 ...
##  $ fat     : int  1 5 1 0 2 2 0 2 1 0 ...
##  $ sodium  : int  130 15 260 140 200 180 125 210 200 210 ...
##  $ fiber   : num  10 2 9 14 1 1.5 1 2 4 5 ...
##  $ carbo   : num  5 8 7 8 14 10.5 11 18 15 13 ...
##  $ sugars  : int  6 8 5 0 8 10 14 8 6 5 ...
##  $ potass  : int  280 135 320 330 -1 70 30 100 125 190 ...
##  $ vitamins: int  25 0 25 25 25 25 25 25 25 25 ...
##  $ weight  : num  1 1 1 1 1 1 1 1.33 1 1 ...
```

```
head(cereal)
```

```
##                         name calories protein fat sodium fiber carbo sugars
## 1                  100% Bran       70       4   1    130  10.0   5.0      6
## 2          100% Natural Bran      120       3   5     15   2.0   8.0      8
## 3                   All-Bran       70       4   1    260   9.0   7.0      5
## 4 All-Bran with Extra Fiber       50       4   0    140  14.0   8.0      0
## 5             Almond Delight      110       2   2    200   1.0  14.0      8
## 6   Apple Cinnamon Cheerios      110       2   2    180   1.5  10.5     10
##   potass vitamins weight
## 1    280       25      1
## 2    135        0      1
## 3    320       25      1
## 4    330       25      1
## 5     -1       25      1
## 6     70       25      1
```

Univariate Analysis

Name

Name is a nominal variable that represents the type of cereal we are evaluating, so that we can consider the ideal bowl of breakfast to start your day with the optimal amount of nutritional property.

Calories

lets Explore the Calories Variable, The amount of energy in an item of food or drink is measured in calories.This is the quantity of calories in 28g.

```
head(cereal$calories)
```

```
## [1]  70 120  70  50 110 110
```

```
length(cereal$calories)
```

```
## [1] 77

table(cereal$calories)

##
##  50  70  80  90 100 110 120 130 140 150 160
##   3   2   1   7  17  29  10   2   3   2   1
```

The table function() returns the total number of specific value (one) that how many times a specific number is present in the calories/dataset.

```
unique(cereal$calories)

##  [1]  70 120  50 110 130  90 100 140 150 160  80

length(unique(cereal$calories))

## [1] 11

min(cereal$calories)

## [1] 50

max(cereal$calories)

## [1] 160
```

Here the total observation of calories is 77 and is a continuous variable that range lies between 50-160. Now, we will see the summary of the calories variable, for further analysis as follows:

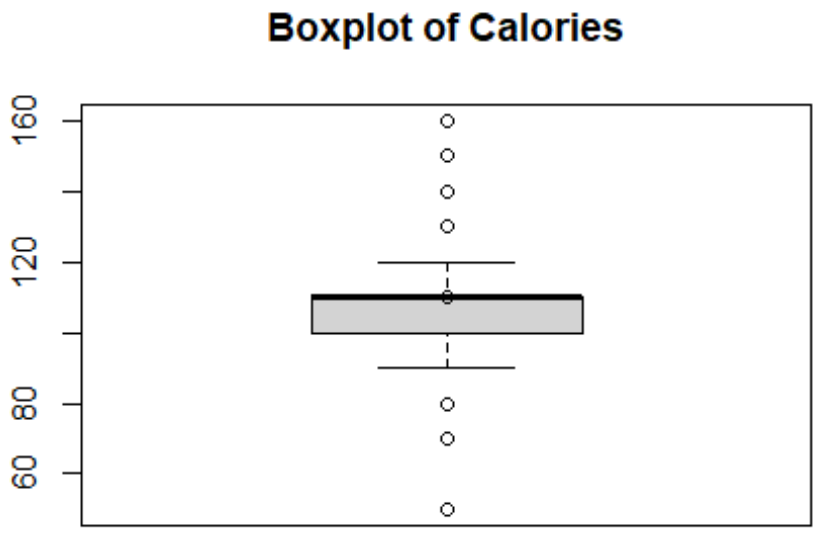
```
summary(cereal$calories)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    50.0   100.0   110.0   106.9   110.0   160.0

sd(cereal$calories)

## [1] 19.48412

var(cereal$calories)

## [1] 379.6309

getMode <- function(v) {
 uniqv <- unique(v)
 uniqv[which.max(tabulate(match(v, uniqv)))]
}
v <- c(cereal$calories)
mode <- getMode(v)
print(mode)

## [1] 110
```

From above statistical computation we can observed that the mean and median are not identical, therefore the distribution is asymmetrical and skewed. And as (Median) is greater than (Mean), so the distribution could be skewed to the left. Now we also confirm this as follows:

```
library(moments)
skewness(cereal$calories)

## [1] -0.4366826
```

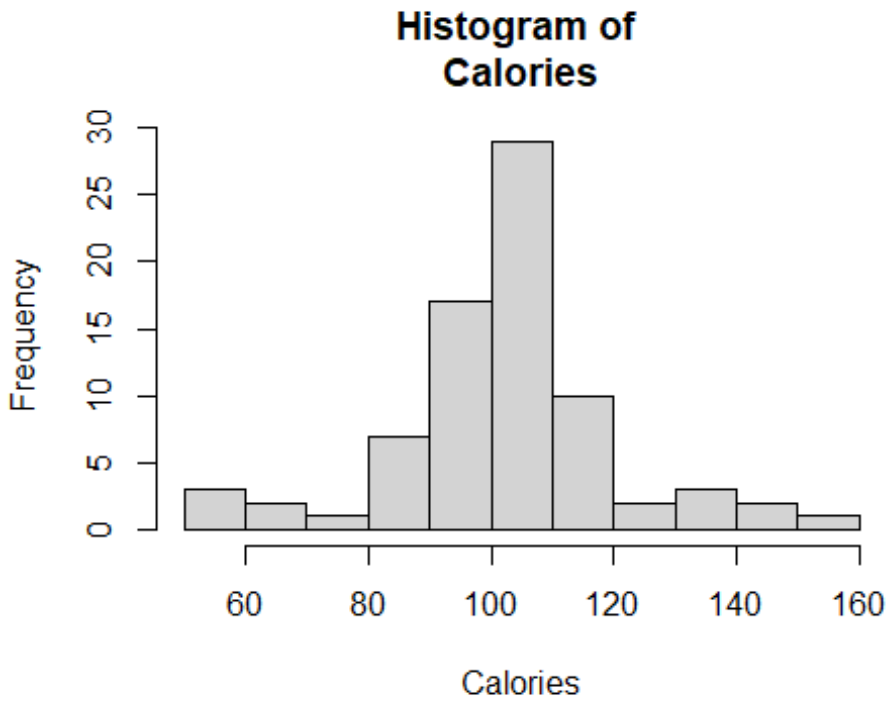
Hence, the skewness is the negative so the distribution is negatively skewed was proved. Also I will plot the Box plot to comparing the distribution of data across dataset as follows:

```
boxplot(cereal$calories, main = "Boxplot of Calories")
points(median(cereal$calories))
```



The boxplot is a graphical representation extremely useful to describe the presence of outliers, in fact the interquartile range that represents 50% of values is in the range 100-110 (cal), there are some outliers out from the range covered by the whiskers as we can see and Now I will plot a histogram a graphical display of data using bars of different heights to measures distribution of continuous data as follows:

```
hist(cereal$calories, breaks = 11, xlab = "Calories", main = "Histogram of
Calories")
```



As our dataset is related to cereal so from above plotted histogram we can observed that the most used calories lies between is 100 to 110. To check whether the distribution is Platykurtic or not, we will check this by following R method as follows:

```
kurtosis(cereal$calories)
```

```
## [1] 5.14209
```

The distribution is also Leptokurtic, since the value is greater than 3. Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution.

Data Fitting for calories:

Now we will try to fit different models to calories distribution evaluating the Akaike Information Criterion (AIC) and the Bayesian Information criterion (BIC), The lower are the indexes, the better is the fitting. I will evaluate both the single parameter distributions and mixture distributions as follows:

```
library(MASS)
```

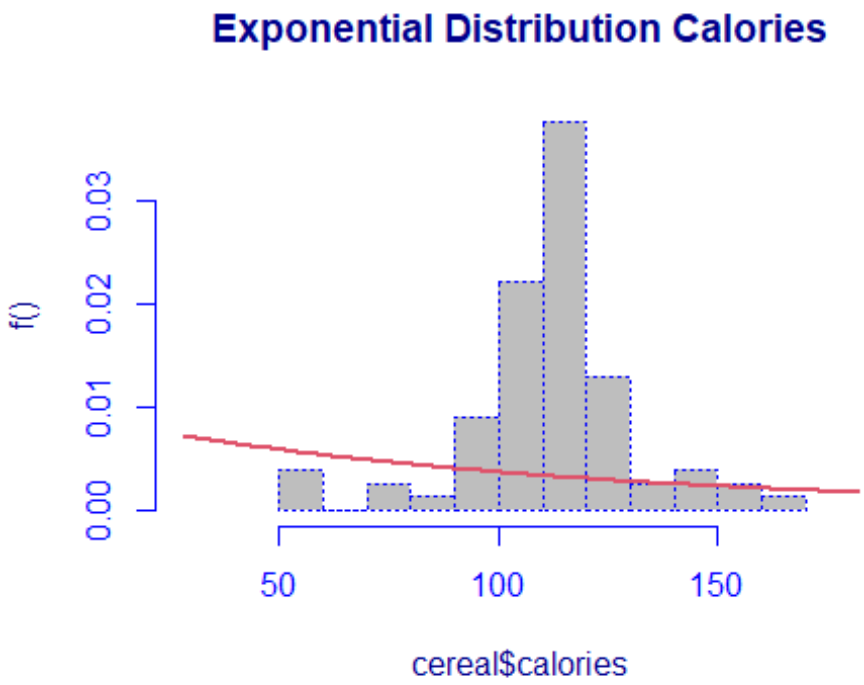
```
library(gamlss)
```

```
## Loading required package: splines
```

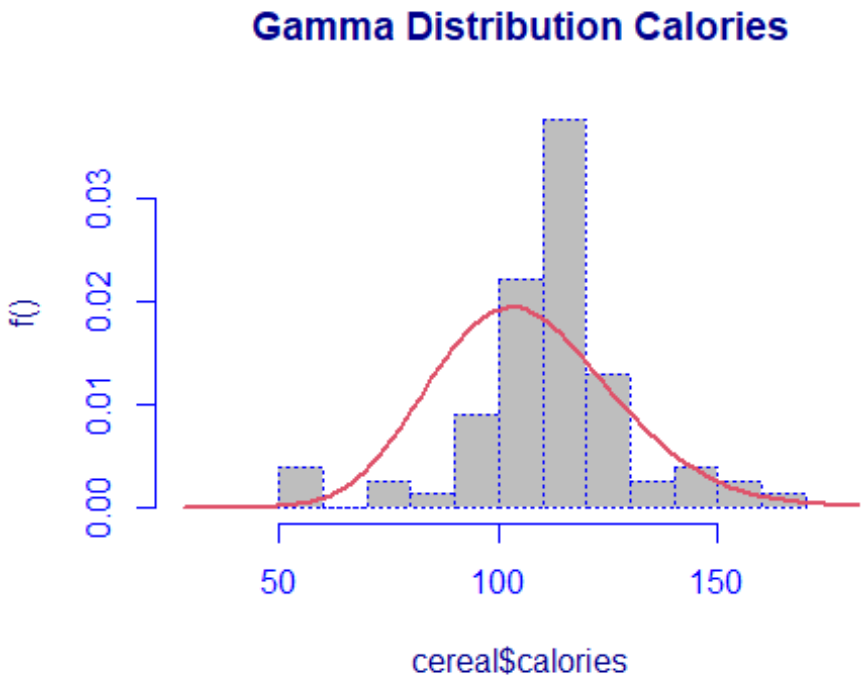
```
## Loading required package: gamlss.data
```

```
##
## Attaching package: 'gamlss.data'
```

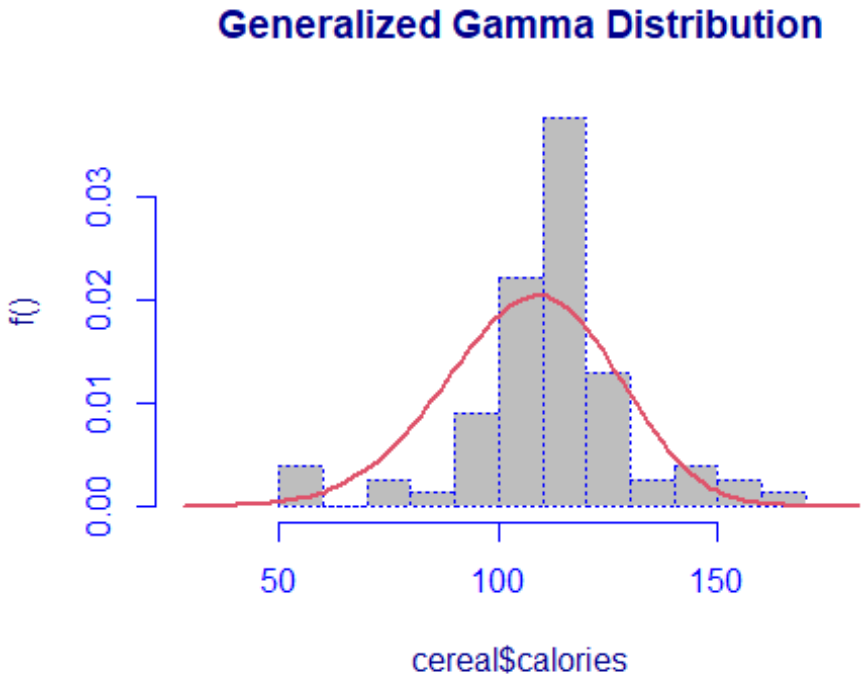
```
## The following object is masked from 'package:datasets':
##
##     sleep

## Loading required package: gamlss.dist

## Loading required package: nlme

## Loading required package: parallel

##  **********   GAMLSS Version 5.4-1  **********

## For more on GAMLSS look at https://www.gamlss.com/

## Type gamlssNews() to see new features/changes/bug fixes.

library(splines)

Calories.Exp <- histDist(cereal$calories,family = EXP, nbins = 11, main =
"Exponential Distribution Calories")
```



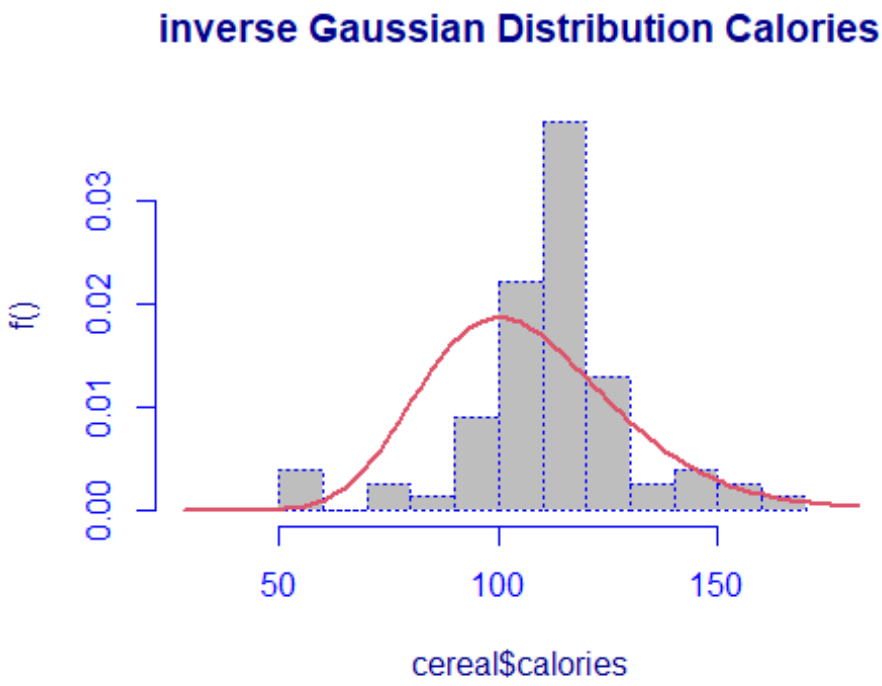

```
Calories.GA <- histDist(cereal$calories,family = GA, nbins = 11, main =
"Gamma Distribution Calories")
```



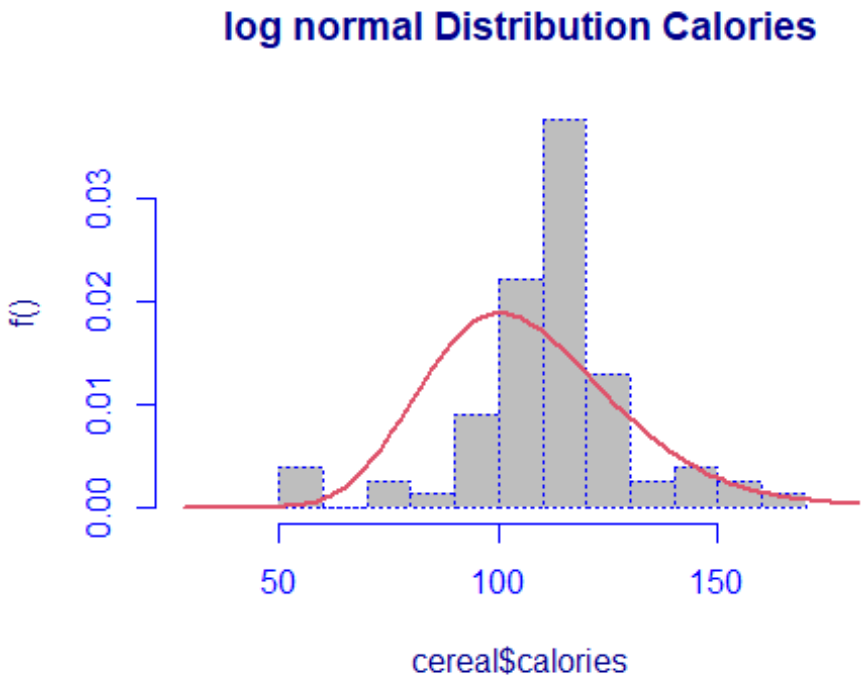

```
Calories.GG <- histDist(cereal$calories,family = GG, nbins = 11, main =
"Generalized Gamma Distribution")
```



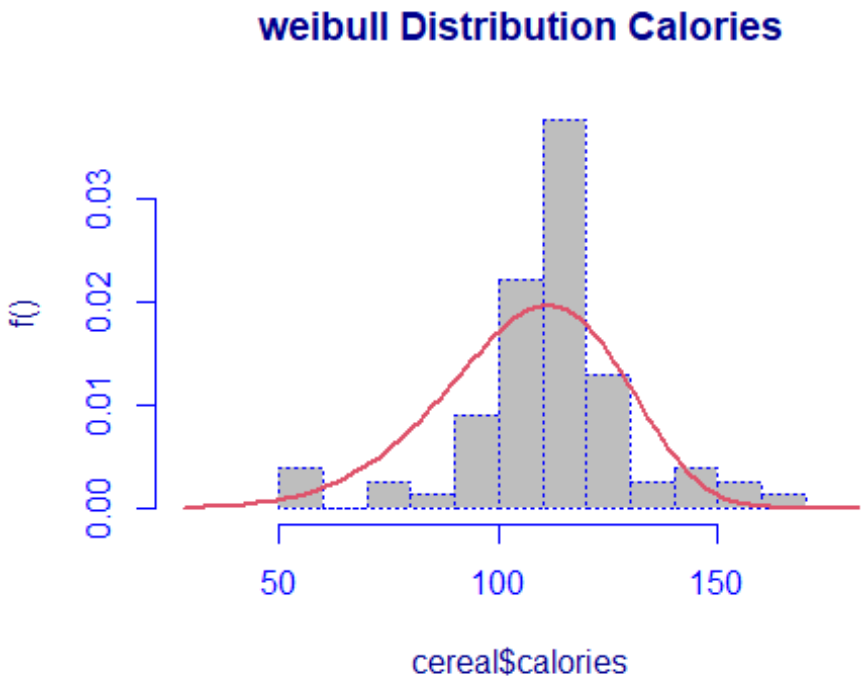
```
Calories.IG <- histDist(cereal$calories,family = IG, nbins = 11, main =
"inverse Gaussian Distribution Calories")
```




```
Calories.LOGNO <- histDist(cereal$calories,family = LOGNO, nbins = 11, main =
"log normal Distribution Calories")
```




```
Calories.WEI <- histDist(cereal$calories,family = WEI, nbins = 11, main =
"weibull Distribution Calories")
```



```
library(Matrix)

data.frame(Row.names = c("Exponential", "Gamma","Generalized Gamma", "Inverse
Gaussian", "Log Normal", "Weibull"),
             AIC = c(AIC(Calories.Exp), AIC(Calories.GA),AIC(Calories.GG),
AIC(Calories.IG), AIC(Calories.LOGNO), AIC(Calories.WEI)),
             BIC = c(Calories.Exp$sbc, Calories.GA$sbc,Calories.GG$sbc,
Calories.IG$sbc, Calories.LOGNO$sbc, Calories.WEI$sbc)
           )

##           Row.names      AIC      BIC
## 1       Exponential 875.4473 877.7911
## 2             Gamma 688.4171 693.1047
## 3 Generalized Gamma 680.9336 687.9650
## 4  Inverse Gaussian 697.2716 701.9593
## 5        Log Normal 695.8427 700.5303
## 6           Weibull 680.5871 685.2747
```

According to the Alaike's An Information Criterion (AIC) and the Bayesian Information Criterion (SBC) for evaluating and comparing statistical model used to establish which model fits better the distribution. In order to evaluate the goodness – of – fit, we use the Likelihood-ratio-test between the Exponential model (under the null hypothesis) and the Weibull model (under the alternative hypothesis):

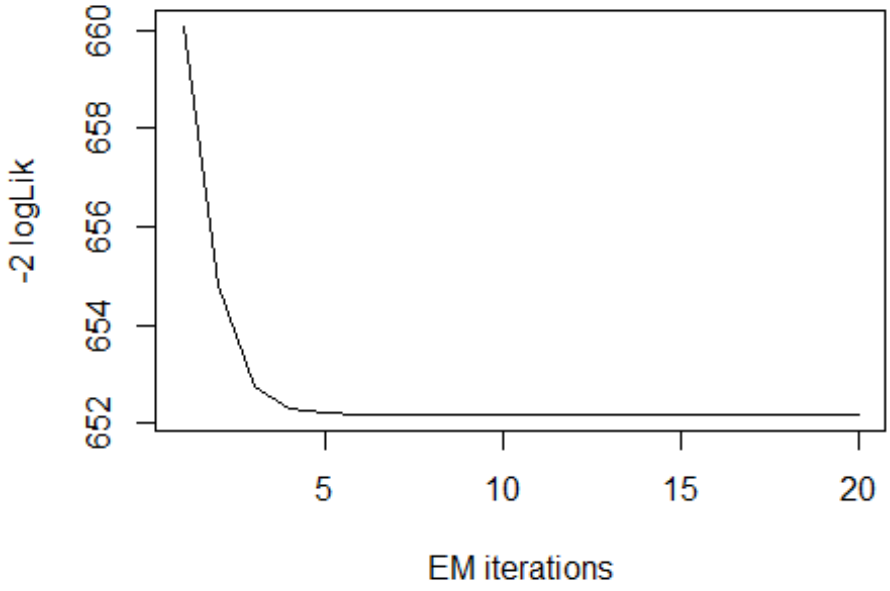
```
LR.test(Calories.Exp,Calories.WEI)

##  Likelihood Ratio Test for nested GAMLSS models.
##  (No check whether the models are nested is performed).
##
##        Null model: deviance= 873.4473 with  1 deg. of freedom
##  Altenative model: deviance= 676.5871 with  2 deg. of freedom
##
##  LRT = 196.8602 with 1 deg. of freedom and p-value= 0
```

As we can see the p-value is 0, so the null model (Exponential model) is rejected, this confirms that the Weibull model fits better our data.

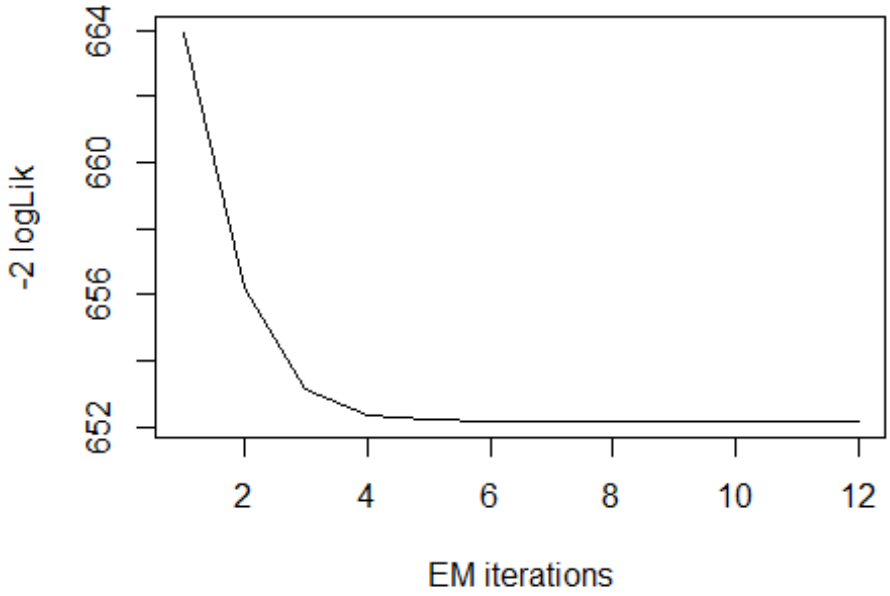
Distributions Mixture

Now, I will also apply the fitting criteria for the distributions with Gamma mixture as follows:

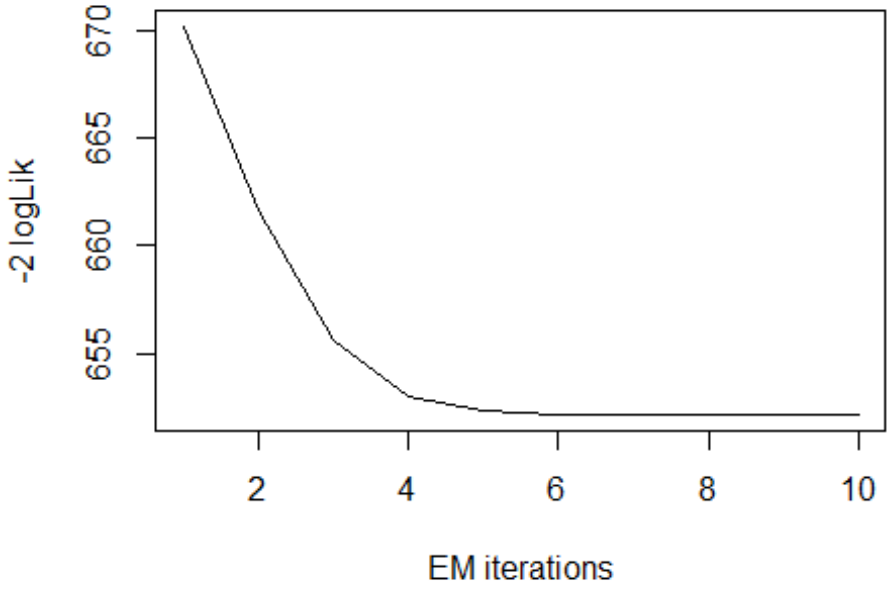
```
library(gamlss.mx)

## Loading required package: nnet

library(nnet)

mix.gam <- gamlssMXfits(n = 6, cereal$calories~1, family = GA, K = 2, data =
cereal)
```



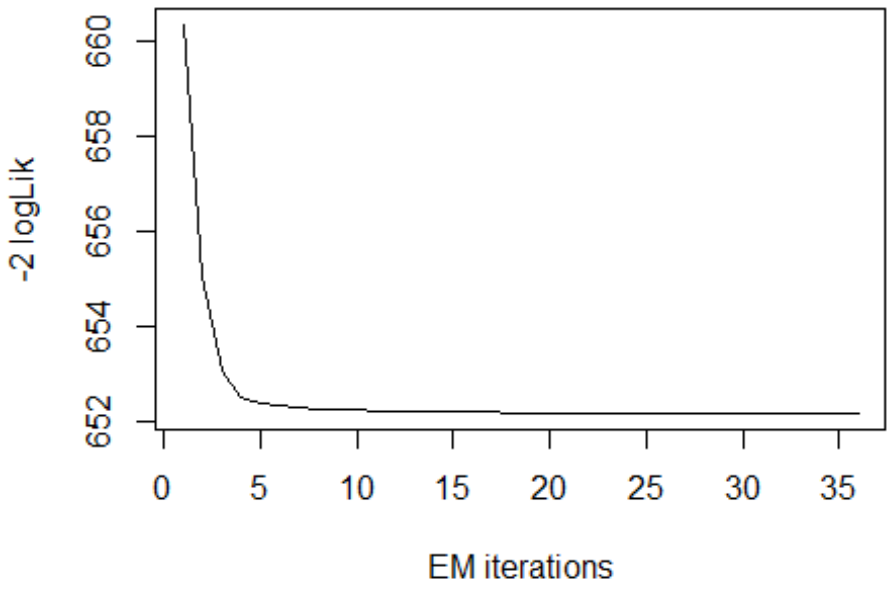

```
## model= 1
```



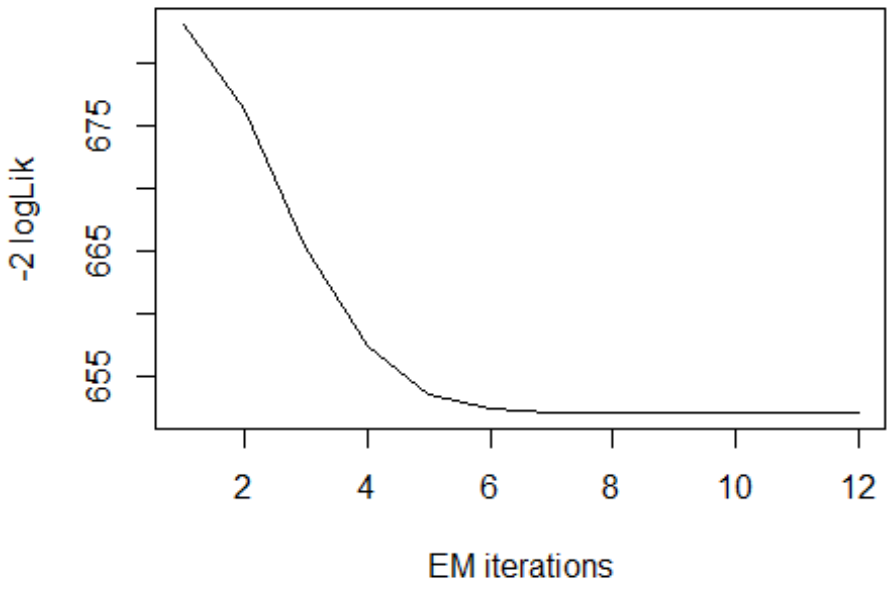

```
## model= 2
```



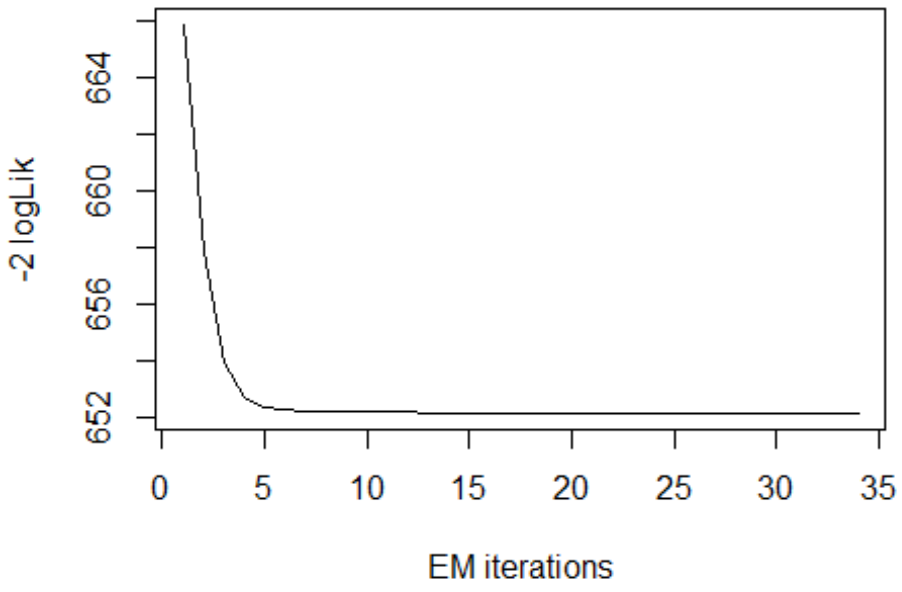

```
## model= 3
```




```
## model= 4
```



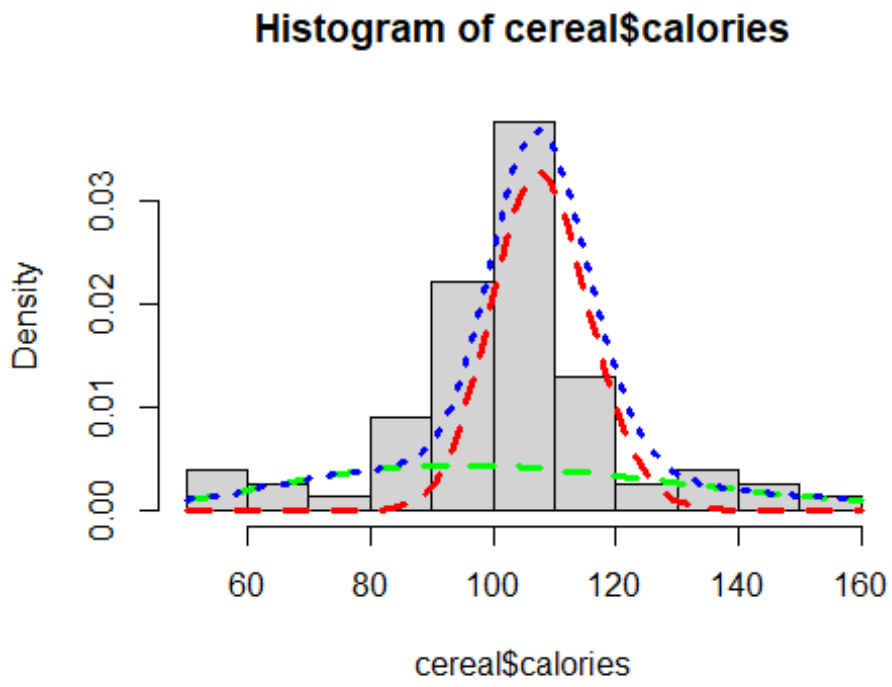

```
## model= 5
```




```
## model= 6
```

```
print(mix.gam)

##
## Mixing Family:  c("GA", "GA")
##
## Fitting method: EM algorithm
##
## Call:  gamlssMX(formula = cereal$calories ~ 1, family = GA,
##     K = 2, data = cereal)
##
## Mu Coefficients for model: 1
## (Intercept)
##       4.682
## Sigma Coefficients for model: 1
## (Intercept)
##      -2.607
## Mu Coefficients for model: 2
## (Intercept)
##       4.652
## Sigma Coefficients for model: 2
## (Intercept)
##       -1.16
##
## Estimated probabilities: 0.6534102 0.3465898
##
## Degrees of Freedom for the fit: 5 Residual Deg. of Freedom   72
## Global Deviance:     652.173
##             AIC:     662.173
##             SBC:     673.892

library(mgcv)

## This is mgcv 1.8-38. For overview type 'help("mgcv-package")'.

##
## Attaching package: 'mgcv'

## The following object is masked from 'package:nnet':
##
##     multinom

 library(MASS)

mu.hat1<-exp(mix.gam[["models"]][[1]][["mu.coefficients"]])

sigma.hat1<-exp(mix.gam [["models"]][[1]][["sigma.coefficients"]])

mu.hat2<-exp(mix.gam [["models"]][[2]][["mu.coefficients"]])

sigma.hat2<-exp(mix.gam[["models"]][[2]][["sigma.coefficients"]])
hist(cereal$calories,breaks = 11,freq = FALSE)
```

```
lines(seq(min(cereal$calories), max(cereal$calories), length =
length(cereal$calories)),
 mix.gam[["prob"]][1]*dGA(seq(min(cereal$calories), max(cereal$calories),
 length = length(cereal$calories)), mu = mu.hat1, sigma = sigma.hat1),
 lty=2, lwd=3, col="red")

lines(seq(min(cereal$calories), max(cereal$calories), length =
length(cereal$calories)),
 mix.gam[["prob"]][2]*dGA(seq(min(cereal$calories), max(cereal$calories),
 length = length(cereal$calories)), mu = mu.hat2, sigma = sigma.hat2),
 lty=2, lwd=3, col="green")


lines(seq(min(cereal$calories), max(cereal$calories), length =
length(cereal$calories)),
 mix.gam[["prob"]][1]*dGA(seq(min(cereal$calories), max(cereal$calories),
length = length(cereal$calories)), mu = mu.hat1, sigma = sigma.hat1)+
mix.gam[["prob"]][2]*dGA(seq(min(cereal$calories), max(cereal$calories),
 length = length(cereal$calories)), mu = mu.hat2, sigma = sigma.hat2),
 lty = 3, lwd = 3, col = "blue")
```



We can see that the values of AIC and SBC have improved, so that Mixture Gamma Distributions fits better the model. The first Gamma distribution, that one with red dot line accounts for 35 %, while the second Gamma distribution accounts for 65%.

Protein

lets Explore the protein Variable,

```
head(cereal$protein)

## [1] 4 3 4 4 2 2

length(cereal$protein)

## [1] 77

table(cereal$protein)

##
##  1  2  3  4  5  6
## 13 25 28  8  1  2
```

The table function() returns the total number of specific value (one) that how many times a specific number is present in the protein/dataset.

```
unique(cereal$protein)

## [1] 4 3 2 1 6 5

length(unique(cereal$protein))

## [1] 6

min(cereal$protein)

## [1] 1

max(cereal$protein)

## [1] 6
```

Here the total observation of protein is 77 and is a continuous variable that range lies between 1-6. Now, we will see the summary of the fat variable, for further analysis as follows:

```
summary(cereal$protein)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   2.545   3.000   6.000

sd(cereal$protein)

## [1] 1.09479

var(cereal$protein)

## [1] 1.198565
```

```
getMode <- function(v) {
 uniqv <- unique(v)
 uniqv[which.max(tabulate(match(v, uniqv)))]
}
v <- c(cereal$protein)
mode <- getMode(v)
print(mode)
```

```
## [1] 3
```

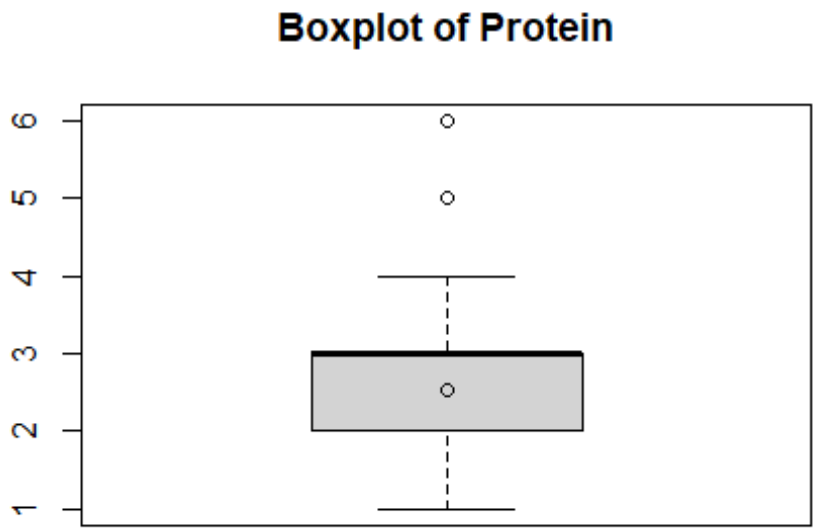
From above statistical computation we can observed that the mean and median are not identical, therefore the distribution is asymmetrical and skewed. And as (Median) is greater than (Mean), so the distribution could be skewed to the right. Now we also confirm this as follows:

```
library(moments)
skewness(cereal$protein)
```

```
## [1] 0.7312214
```

Hence, the skewness is the positive so the distribution is positively skewed was proved. Also I will plot the Box plot to comparing the distribution of data across dataset as follows:

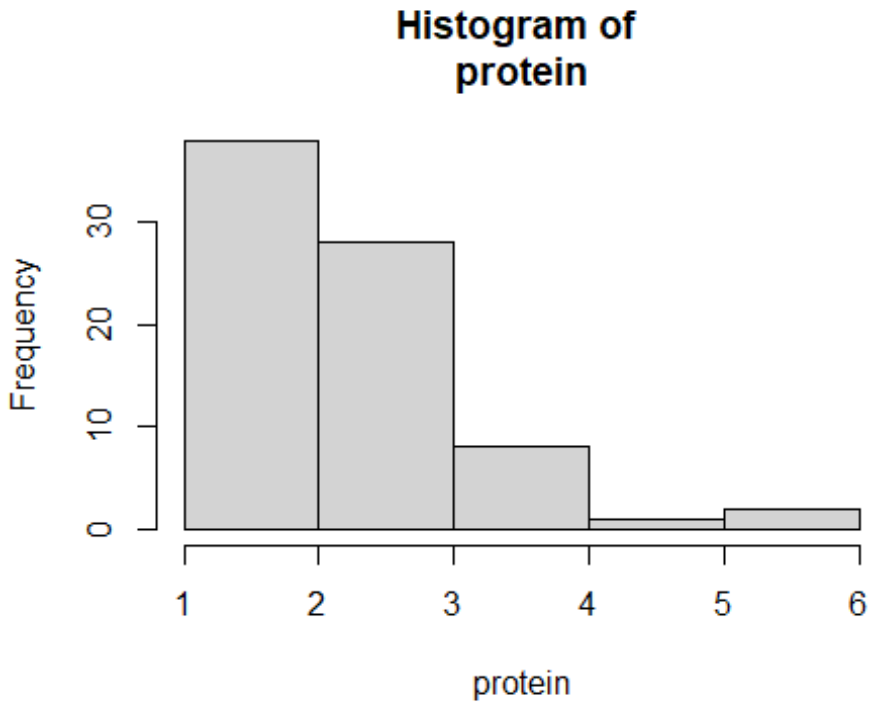
```
boxplot(cereal$protein, main = "Boxplot of Protein")
points(mean(cereal$protein))
```



Box plots are graphical displays of the five number summary, So they describe the Minimum (lowest mean relative) lower whisker, Quartile 1 (It separates the lowest 25% observation to the

rest of the observations), Median (Which divides the lower 50% observation from the upper 50% observations), Quartile 3 (It divides the upper 25% observation to the rest of the observations) and Maximum. In the above Box plot diagram, there are no outliers and the upper whisker shows the maximum. Now, I will plot a histogram a graphical display of data using bars of different heights to measures distribution of continuous data as follows:

```
hist(cereal$protein, breaks = 6, xlab = "protein", main = "Histogram of
protein")
```



From the above histogram, we can observe that the diagram has many peaks.Using the density lines, I have approximated the histogram graph. Density provides information about the type of distribution under consideration and allows us to determine whether the attribute is distributed normally or not. From the graph above we can see how the Age variable does not seem to fit to a Normal distribution, there is a peak of the frequency distributions around many values and also a left skewed. We have already observed a positive value of 0.7312214, so we will affirm that the distribution is right skewed. To check whether the distribution is Platykurtic or not, we will check this by following R method as follows:

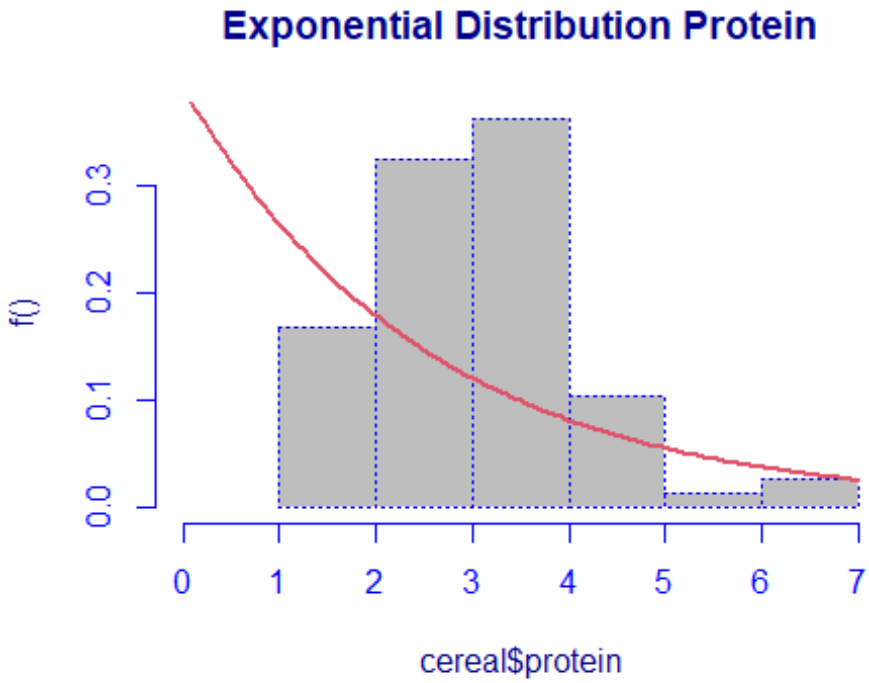
```
kurtosis(cereal$protein)

## [1] 4.032193
```

The distribution is also Leptokurtic, since the value is greater than 3. Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution.

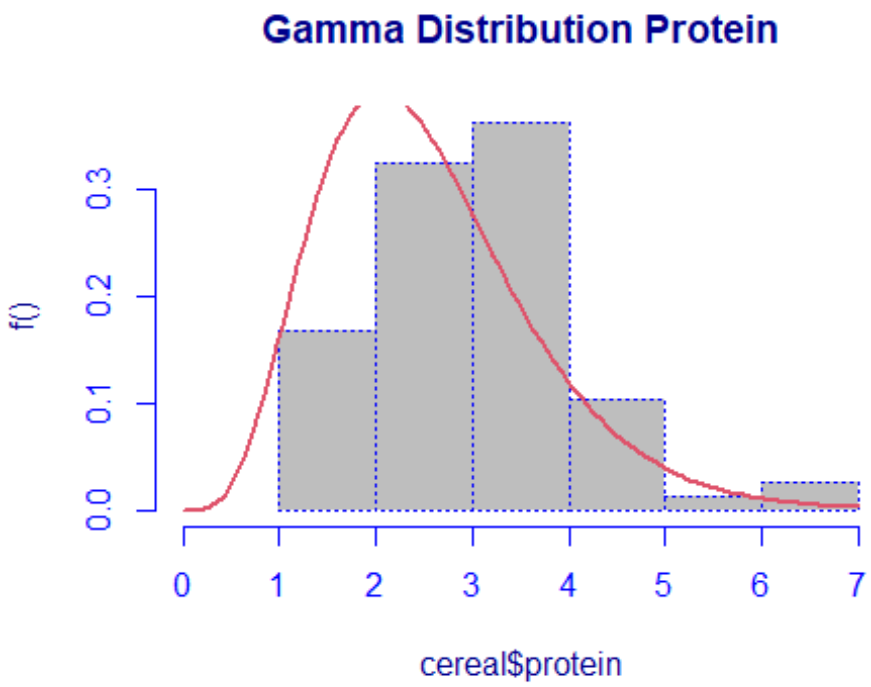
Data Fitting for fat:

Now we will try to fit different models to fat distribution evaluating the Akaike Information Criterion (AIC) and the Bayesian Information criterion (BIC), The lower are the indexes, the better is the fitting. I will evaluate both the single parameter distributions and mixture distributions as follows:

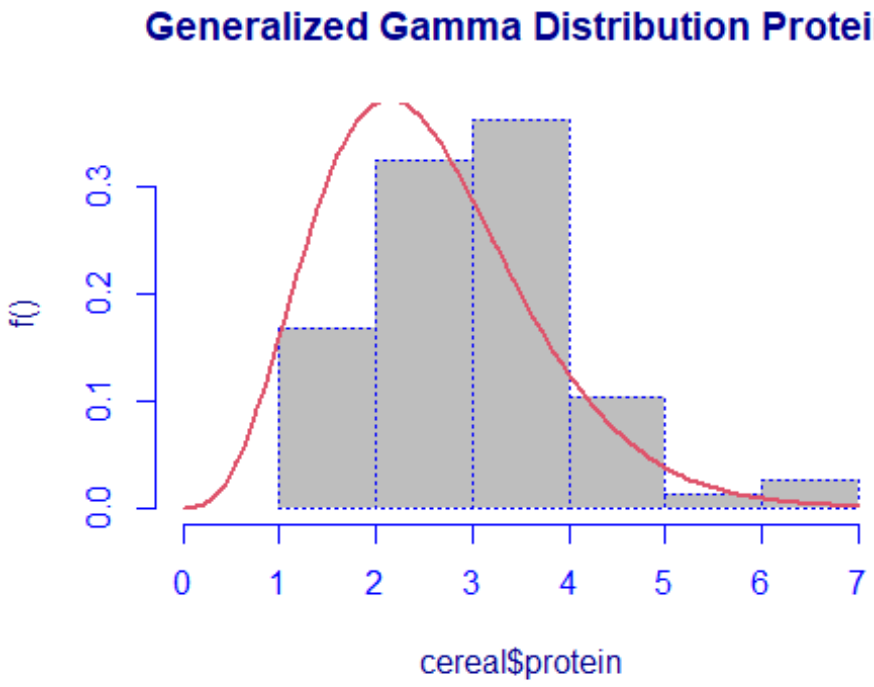
```
library(MASS)

library(gamlss)

library(splines)

protein.Exp <- histDist(cereal$protein,family = EXP, nbins = 6, main =
"Exponential Distribution Protein")
```



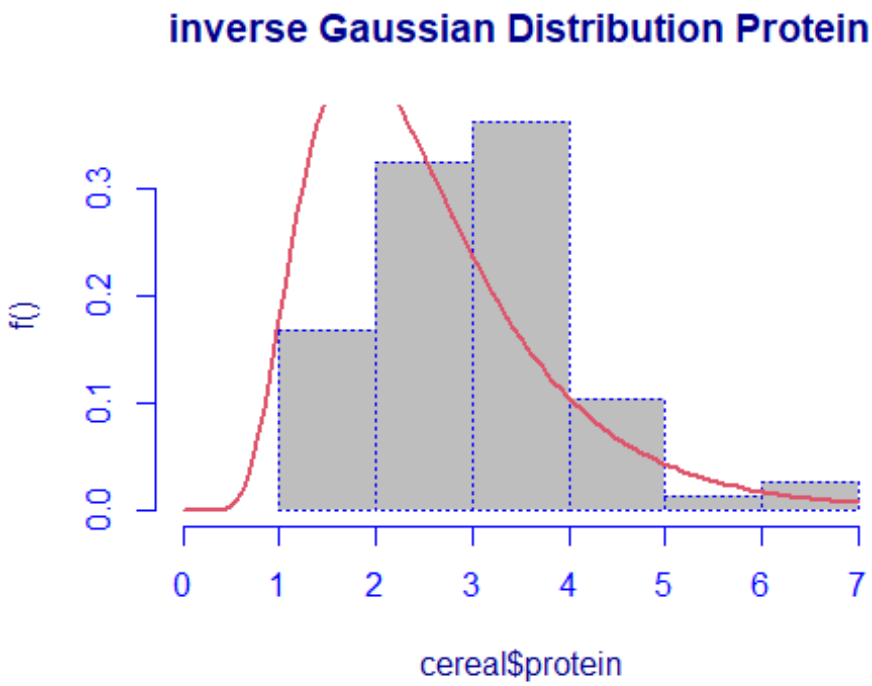

```
protein.GA <- histDist(cereal$protein,family = GA, nbins = 6, main = "Gamma
Distribution Protein")
```



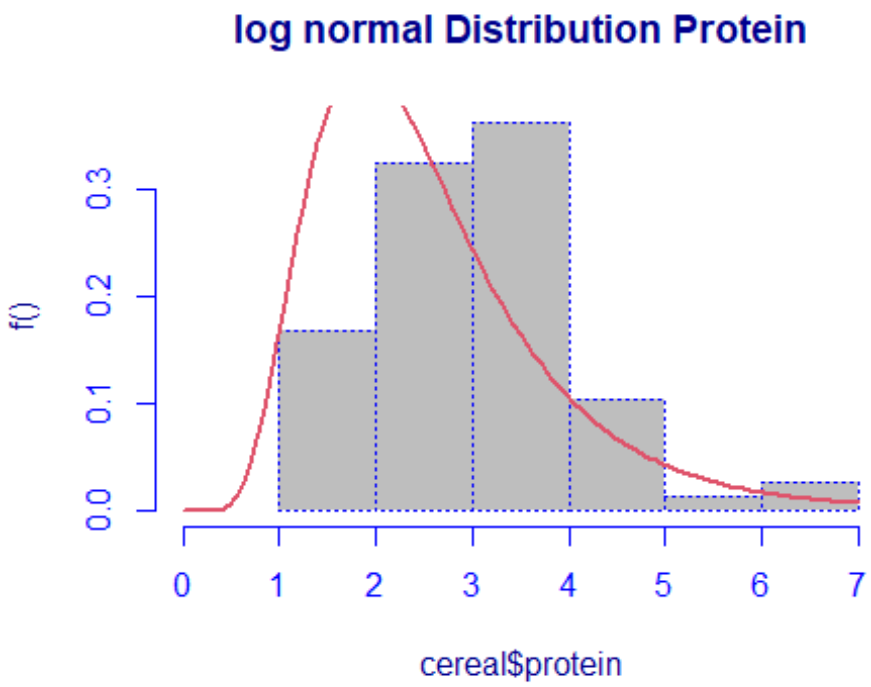

```
protein.GG <- histDist(cereal$protein,family = GG, nbins = 6, main =
"Generalized Gamma Distribution Protein")
```



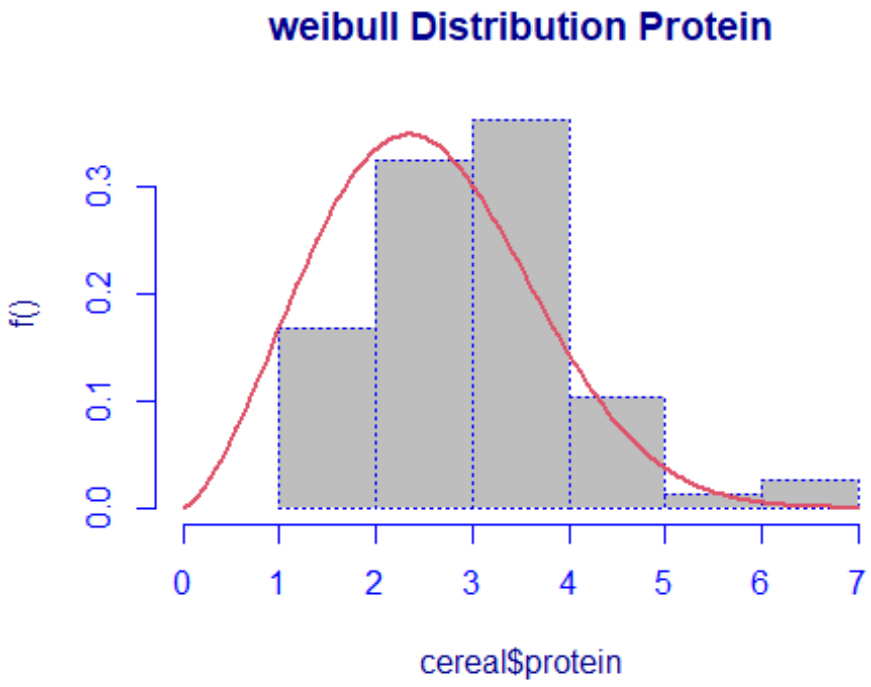
```
protein.IG <- histDist(cereal$protein,family = IG, nbins = 6, main = "inverse
Gaussian Distribution Protein")
```




```
protein.LOGNO <- histDist(cereal$protein,family = LOGNO, nbins = 6, main =
"log normal Distribution Protein")
```




```
protein.WEI <- histDist(cereal$protein,family = WEI, nbins = 6, main =
"weibull Distribution Protein")
```



```
library(Matrix)

data.frame(Row.names = c("Exponential", "Gamma","Generalized Gamma", "Inverse
Gaussian", "Log Normal", "Weibull"),
             AIC = c(AIC(protein.Exp), AIC(protein.GA),AIC(protein.GG),
AIC(protein.IG), AIC(protein.LOGNO), AIC(protein.WEI)),
             BIC = c(protein.Exp$sbc, protein.GA$sbc,protein.GG$sbc,
protein.IG$sbc, protein.LOGNO$sbc, protein.WEI$sbc)
           )

##           Row.names      AIC      BIC
## 1       Exponential 299.8836 302.2274
## 2             Gamma 228.4919 233.1795
## 3 Generalized Gamma 230.2159 237.2473
## 4  Inverse Gaussian 232.2708 236.9584
## 5        Log Normal 232.0611 236.7487
## 6           Weibull 230.3480 235.0356
```

According to the Alaike's An Information Criterion (AIC) and the Bayesian Information Criterion (SBC) for evaluating and comparing statistical model used to establish which model fits better the distribution. In order to evaluate the goodness – of – fit, we use the Likelihood-ratio-test between the Exponential model (under the null hypothesis) and the Weibull model (under the alternative hypothesis):

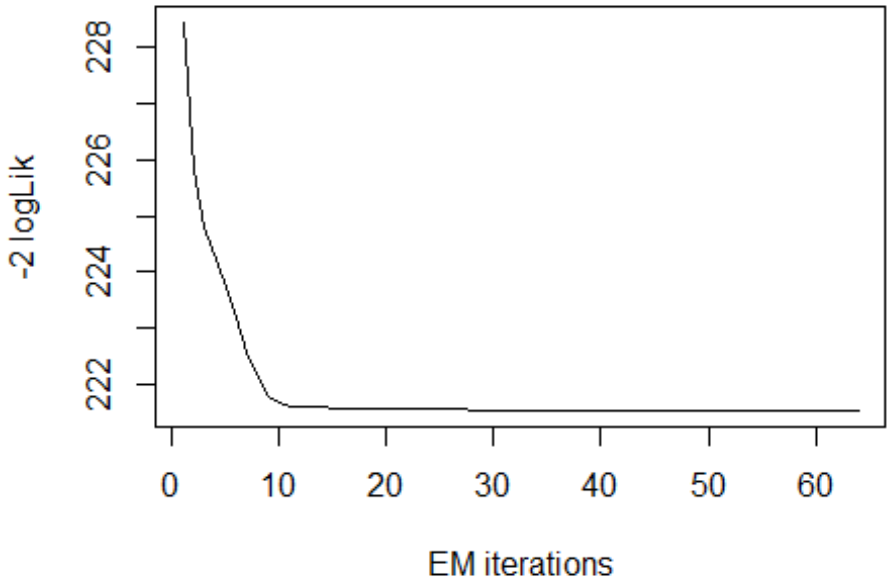
```
LR.test(protein.Exp,protein.GA)

##  Likelihood Ratio Test for nested GAMLSS models.
##  (No check whether the models are nested is performed).
##
##        Null model: deviance= 297.8836 with  1 deg. of freedom
##  Altenative model: deviance= 224.4919 with  2 deg. of freedom
##
##  LRT = 73.39177 with 1 deg. of freedom and p-value= 0
```

As we can see the p-value is 0, so the null model (Exponential model) is rejected, this confirms that the GAMMA model fits better our data.

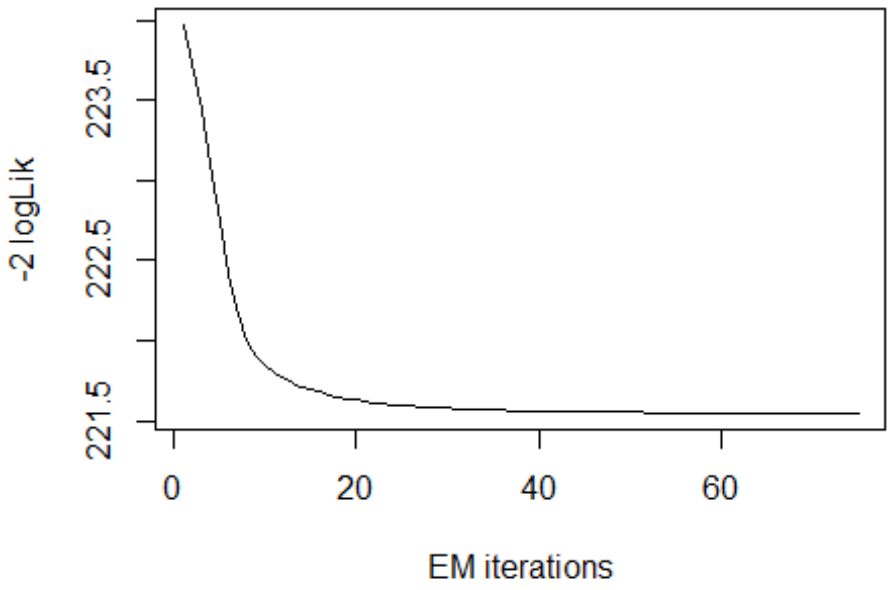
Distributions Mixture

Now, I will also apply the fitting criteria for the distributions with Gamma mixture as follows:

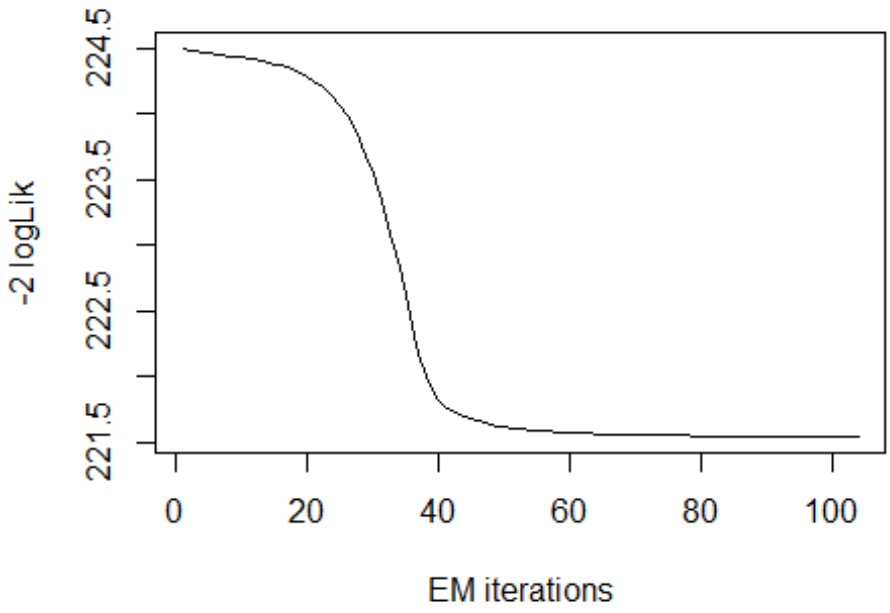
```
library(gamlss.mx)

library(nnet)

mix.gam <- gamlssMXfits(n = 6, cereal$protein~1, family = GA, K = 2, data =
cereal)
```



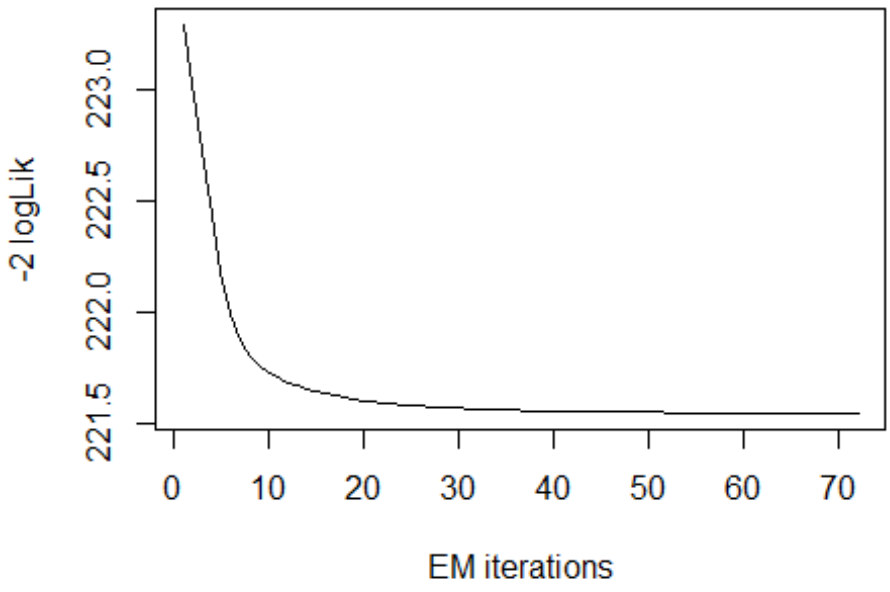

```
## model= 1
```



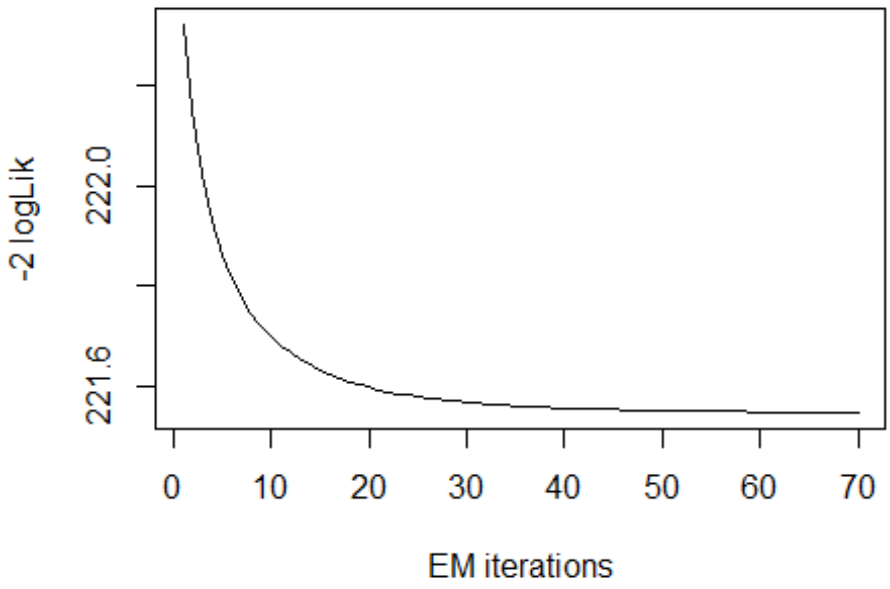

```
## model= 2
```



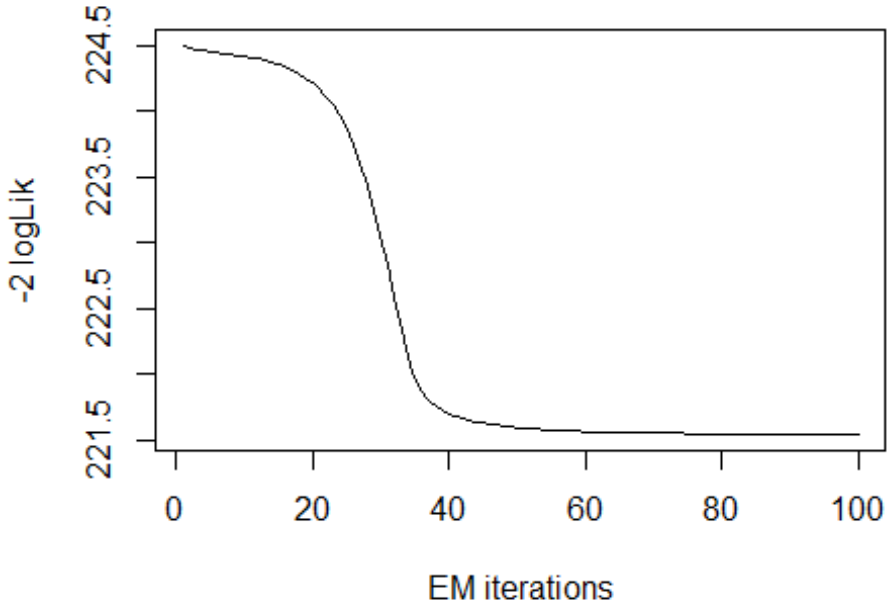

```
## model= 3
```




```
## model= 4
```



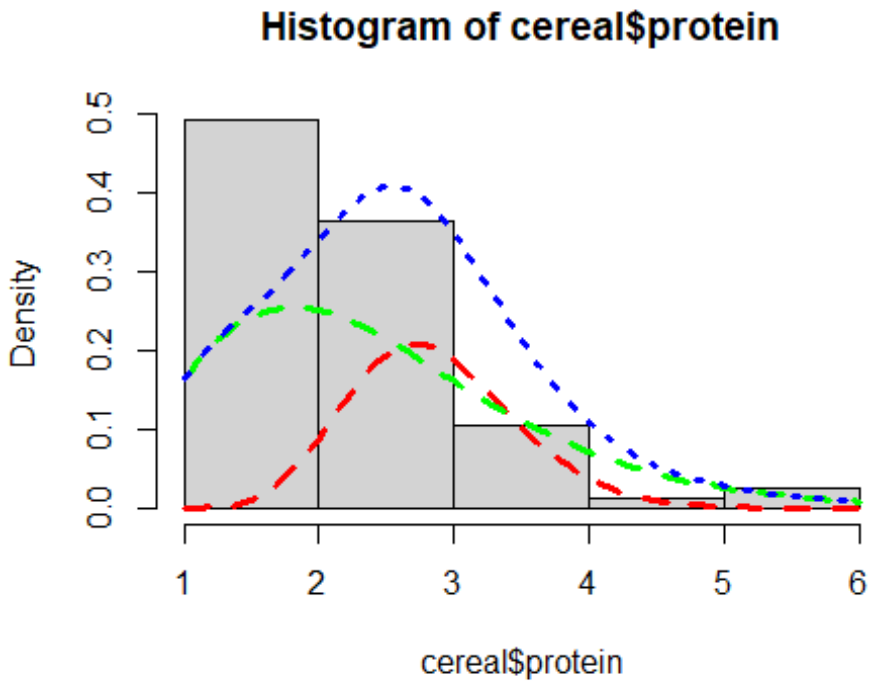

```
## model= 5
```




```
## model= 6
```

```
print(mix.gam)

##
## Mixing Family:  c("GA", "GA")
##
## Fitting method: EM algorithm
##
## Call:  gamlssMX(formula = cereal$protein ~ 1, family = GA,
##     K = 2, data = cereal)
##
## Mu Coefficients for model: 1
## (Intercept)
##       1.048
## Sigma Coefficients for model: 1
## (Intercept)
##      -1.524
## Mu Coefficients for model: 2
## (Intercept)
##      0.8764
## Sigma Coefficients for model: 2
## (Intercept)
##     -0.7003
##
## Estimated probabilities: 0.3179924 0.6820076
##
## Degrees of Freedom for the fit: 5 Residual Deg. of Freedom   72
## Global Deviance:     221.547
##             AIC:     231.547
##             SBC:     243.266

library(mgcv)

 library(MASS)

mu.hat1<-exp(mix.gam[["models"]][[1]][["mu.coefficients"]])

sigma.hat1<-exp(mix.gam [["models"]][[1]][["sigma.coefficients"]])

mu.hat2<-exp(mix.gam [["models"]][[2]][["mu.coefficients"]])

sigma.hat2<-exp(mix.gam[["models"]][[2]][["sigma.coefficients"]])
hist(cereal$protein,breaks = 6,freq = FALSE)


lines(seq(min(cereal$protein), max(cereal$protein), length =
length(cereal$protein)),
 mix.gam[["prob"]][1]*dGA(seq(min(cereal$protein), max(cereal$protein),
 length = length(cereal$protein)), mu = mu.hat1, sigma = sigma.hat1),
 lty=2, lwd=3, col="red")
```

```
lines(seq(min(cereal$protein), max(cereal$protein), length =
length(cereal$protein)),
 mix.gam[["prob"]][2]*dGA(seq(min(cereal$protein), max(cereal$protein),
 length = length(cereal$protein)), mu = mu.hat2, sigma = sigma.hat2),
 lty=2, lwd=3, col="green")


lines(seq(min(cereal$protein), max(cereal$protein), length =
length(cereal$protein)),
 mix.gam[["prob"]][1]*dGA(seq(min(cereal$protein), max(cereal$protein),
length = length(cereal$protein)), mu = mu.hat1, sigma = sigma.hat1)+
mix.gam[["prob"]][2]*dGA(seq(min(cereal$protein), max(cereal$protein),
 length = length(cereal$protein)), mu = mu.hat2, sigma = sigma.hat2),
 lty = 3, lwd = 3, col = "blue")
```



We can see that the values of AIC and SBC have improved, so that Mixture Gamma Distributions fits better the model. The first Gamma distribution, that one with red dot line accounts for 32 %, while the second Gamma distribution accounts for 68%.

Principal Component Analysis

Principal Component Analysis is a statistical procedure to convert a set of n observations of possibly correlated numerical variables into a set of n values of new linearly uncorrelated variables. The goal of PCA is to explain the variability in the data with a new smaller number of variables than the original data set. Firstly, we have to understand if using PCA can be useful, so we have to analyze if there is correlation between variables.

```
library(psych)
```

```
## Warning: package 'psych' was built under R version 4.1.3
```

```
##
## Attaching package: 'psych'
```

```
## The following object is masked from 'package:gamlss':
##
##     cs
```

```
cereal_sub<-cereal[,-c(1)]
cor(cereal_sub)
```

```
##             calories      protein          fat       sodium       fiber
## calories  1.00000000  0.019066068  0.498609814  0.300649227 -0.29341275
## protein   0.01906607  1.000000000  0.208430990 -0.054674348  0.50033004
## fat       0.49860981  0.208430990  1.000000000 -0.005407464  0.01671924
## sodium    0.30064923 -0.054674348 -0.005407464  1.000000000 -0.07067501
## fiber    -0.29341275  0.500330043  0.016719237 -0.070675009  1.00000000
## carbo     0.25068091 -0.130863648 -0.318043492  0.355983473 -0.35608274
## sugars    0.56234029 -0.329141777  0.270819175  0.101451381 -0.14120539
## potass   -0.06660886  0.549407400  0.193278602 -0.032603467  0.90337367
## vitamins  0.26535630  0.007335371 -0.031156266  0.361476688 -0.03224268
## weight    0.69609108  0.216158486  0.214625033  0.308576451  0.24722563
##               carbo      sugars      potass     vitamins    weight
## calories  0.2506809  0.56234029 -0.06660886  0.265356298 0.6960911
## protein  -0.1308636 -0.32914178  0.54940740  0.007335371 0.2161585
## fat      -0.3180435  0.27081918  0.19327860 -0.031156266 0.2146250
## sodium    0.3559835  0.10145138 -0.03260347  0.361476688 0.3085765
## fiber    -0.3560827 -0.14120539  0.90337367 -0.032242679 0.2472256
## carbo     1.0000000 -0.33166538 -0.34968522  0.258147549 0.1351364
## sugars   -0.3316654  1.00000000  0.02169581  0.125137260 0.4506476
## potass   -0.3496852  0.02169581  1.00000000  0.020698687 0.4163032
## vitamins  0.2581475  0.12513726  0.02069869  1.000000000 0.3203241
## weight    0.1351364  0.45064760  0.41630315  0.320324059 1.0000000
```

We can see, for example, that there is a high positive correlation between Fiber and Potass, so we perform PCA using the prcomp function to scale the data, so that the standard deviation is one and to compare the variables.

```
pc.out=prcomp(cereal[2:11],scale = TRUE)
pc.out
```

```
## Standard deviations (1, .., p=10):
##  [1] 1.6469960 1.6263407 1.2910785 1.0009107 0.8360410 0.7987013 0.6139803
##  [8] 0.3781165 0.2583268 0.2239408
##
## Rotation (n x k) = (10 x 10):
##                  PC1        PC2        PC3         PC4         PC5
PC6
```

```
## calories -0.22191652 -0.5170331 -0.1066145  0.27571150 -0.19303975 -
0.08497454
## protein  -0.36418410  0.2170508  0.2354390  0.48930132  0.06430679 -
0.13827389
## fat      -0.29289085 -0.1405261 -0.3776753  0.51623948  0.40207804
0.15621675
## sodium   -0.06141487 -0.3252981  0.3704927 -0.13088239  0.28678337
0.78421593
## fiber    -0.43237299  0.3443569  0.1745903 -0.24212053 -0.06160932
0.09367515
## carbo     0.22016419 -0.2662785  0.5215005  0.24438654 -0.35001836 -
0.05623390
## sugars   -0.18506158 -0.3488030 -0.4455845 -0.42956553 -0.08760288 -
0.01614176
## potass   -0.51947123  0.2369929  0.1142316 -0.15802901 -0.07951675
0.05654864
## vitamins -0.10114653 -0.2878746  0.3517955 -0.25615197  0.63745896 -
0.55314330
## weight   -0.42286963 -0.3313796  0.1244959 -0.08644833 -0.40991361 -
0.11357049
##                  PC7          PC8          PC9        PC10
## calories  0.02683329 -0.167458896  0.431910933  0.57987220
## protein   0.65397415 -0.188726690 -0.032833068 -0.19436712
## fat      -0.45068571  0.048495246 -0.092125327 -0.29205925
## sodium    0.19241781  0.025752040 -0.008307845  0.03190915
## fiber    -0.26837441 -0.076992914  0.678564373 -0.23322677
## carbo    -0.36061943 -0.418668347 -0.098755161 -0.32890523
## sugars    0.23957791 -0.490946432 -0.078650116 -0.38531796
## potass   -0.25832519 -0.295507918 -0.542912324  0.42146258
## vitamins -0.06889856  0.003094738  0.012829562  0.01676471
## weight    0.02253582  0.650961441 -0.180312007 -0.22516171
```

From the output above we can put in evidence:

The values below represent the standard deviation of the original variables in the original space.

```
pc.out$scale

##   calories    protein        fat     sodium      fiber      carbo
sugars
## 19.4841191  1.0947897  1.0064726 83.8322952  2.3833640  4.2789563
4.4448854
##     potass   vitamins     weight
## 71.2868125 22.3425225  0.1504768
```

The above is Standard Derivations(SD) of Variables.

```
head(pc.out$x)
```

```
##               PC1        PC2        PC3         PC4       PC5         PC6
## [1,] -3.15975623  3.9060806  0.1050956 -1.44194002 0.7273966  0.38408949
## [2,] -1.78417281  0.6093253 -3.4396609  2.44868683 0.7482849 -0.01980342
## [3,] -3.22052238  3.3441457  1.0144652 -1.42111597 1.0094565  1.56995898
## [4,] -3.33412058  5.5659293  1.9742163 -2.01947800 0.2739453  0.58888867
## [5,]  0.76786021 -0.9146768 -0.8229487  0.49179195 0.6230453  0.57009765
## [6,] -0.08892979 -0.4679303 -1.3879968 -0.07835317 0.7093876  0.49771286
##               PC7         PC8         PC9        PC10
## [1,] -0.02969780 -0.04708785  0.24801927 -0.18453177
## [2,] -1.25496627  0.19458474 -0.26070680 -0.22057975
## [3,]  0.01388627 -0.22589812 -0.38267199  0.13225116
## [4,] -0.79433188  0.11239046  0.69018413 -0.29205358
## [5,] -0.07542796  0.37766814  0.43080711 -0.54833457
## [6,] -0.03215046  0.18260181  0.07980385 -0.08945408
```

The x matrix provides the principal component scores (the projection of the n data points, i.e. rows of X, onto the first eigenvector PC1), while the Rotation matrix of the PCA function contains the principal components loadings (loading vectors are the directions in the original variable space along which the data vary the most). By default, eigenvectors in R point in the negative direction.

```
pc.out$rotation=-pc.out$rotation
pc.out$rotation
```

```
##                  PC1        PC2        PC3         PC4         PC5
PC6
## calories  0.22191652  0.5170331  0.1066145 -0.27571150  0.19303975
0.08497454
## protein   0.36418410 -0.2170508 -0.2354390 -0.48930132 -0.06430679
0.13827389
## fat       0.29289085  0.1405261  0.3776753 -0.51623948 -0.40207804 -
0.15621675
## sodium    0.06141487  0.3252981 -0.3704927  0.13088239 -0.28678337 -
0.78421593
## fiber     0.43237299 -0.3443569 -0.1745903  0.24212053  0.06160932 -
0.09367515
## carbo    -0.22016419  0.2662785 -0.5215005 -0.24438654  0.35001836
0.05623390
## sugars    0.18506158  0.3488030  0.4455845  0.42956553  0.08760288
0.01614176
## potass    0.51947123 -0.2369929 -0.1142316  0.15802901  0.07951675 -
0.05654864
## vitamins  0.10114653  0.2878746 -0.3517955  0.25615197 -0.63745896
0.55314330
## weight    0.42286963  0.3313796 -0.1244959  0.08644833  0.40991361
0.11357049
##                  PC7          PC8          PC9        PC10
## calories -0.02683329  0.167458896 -0.431910933 -0.57987220
## protein  -0.65397415  0.188726690  0.032833068  0.19436712
## fat       0.45068571 -0.048495246  0.092125327  0.29205925
```

```
## sodium   -0.19241781 -0.025752040  0.008307845 -0.03190915
## fiber     0.26837441  0.076992914 -0.678564373  0.23322677
## carbo     0.36061943  0.418668347  0.098755161  0.32890523
## sugars   -0.23957791  0.490946432  0.078650116  0.38531796
## potass    0.25832519  0.295507918  0.542912324 -0.42146258
## vitamins  0.06889856 -0.003094738 -0.012829562 -0.01676471
## weight   -0.02253582 -0.650961441  0.180312007  0.22516171
```

Looking at the first PC, we can see that places most of its weight on potass (0.519), fiber (0.432) with a less weight on protein (0.364). So PC1 roughly corresponds to a measure of those 3 variables. We can also see the presence of variables in particular carbo with a weight equals to (0.22) that has a negative contributes. The second principal component places most if its weight on calories (0.517) and other 2 components, sodium and sugar that have equal weight between 0.32/0.34.

##The proportion of Variance Explained PCA reduces the dimensionality while explaining most of the variability, but there is a more technical method for measuring exactly what percentage of the variance was retained in these principal components.

Proportion of variance" is a generic term to mean a part of variance as a whole. For example, the total variance in any system is 100%, but there might be many different causes for the total variance — each of which have their own proportion associated with them.

We can evaluate the optimal number of components using the Cumulative Proportion of Variance that retain as many PCs as needed to explain at least the 80% of the total variance. The proportion of variance explained by the mth principal component, say PVE:

```
PVE<-pc.out$sdev^2/sum(pc.out$sdev^2)
PVE

##  [1] 0.271259568 0.264498398 0.166688369 0.100182221 0.069896455
0.063792379
##  [7] 0.037697179 0.014297211 0.006673271 0.005014949
```

• The First PC explains 27.12% of the variability. • The Second PC explains 26.4% of the variability. • The Third PC explains 16.66% of the variability. • The Fourth PC explains 10.01% of the variability. • The Fifth PC explains 6.98% of the variability. • The Sixth PC explains 6.37% of the variability. • The Seven PC explains 3.76% of the variability. • The Eight PC explains 1.42% of the variability. • The Ninth PC explains 0.06% of the variability. • The Tenth PC explains 0.05% of the variability.

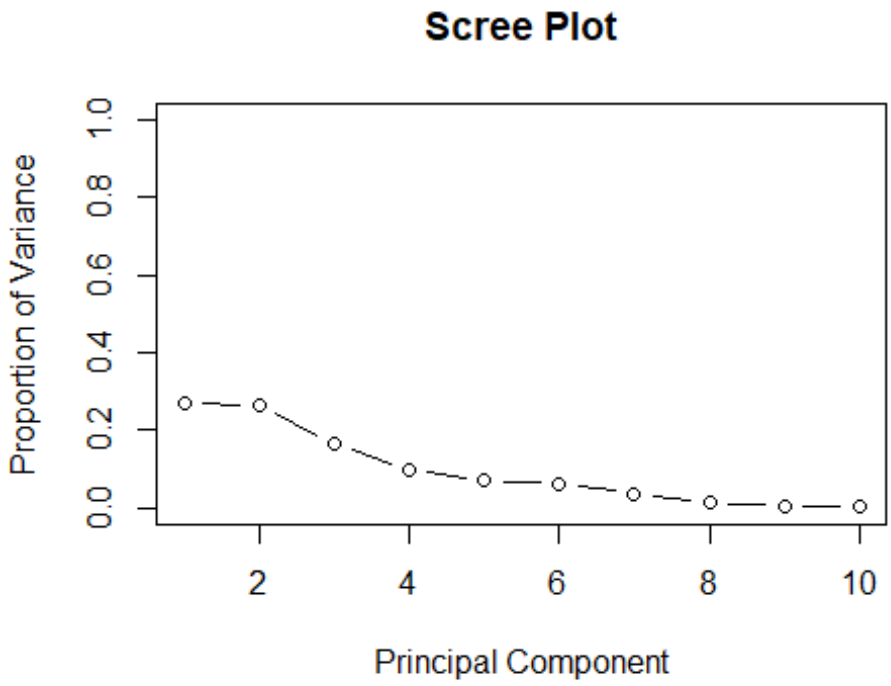
```
cumsum(PVE)

##  [1] 0.2712596 0.5357580 0.7024463 0.8026286 0.8725250 0.9363174 0.9740146
##  [8] 0.9883118 0.9949851 1.0000000
```

This method suggest to retain 4 Principal Components, because the sum of the Eigenvalues of the first four PCs overcomes the threshold of 0.80.

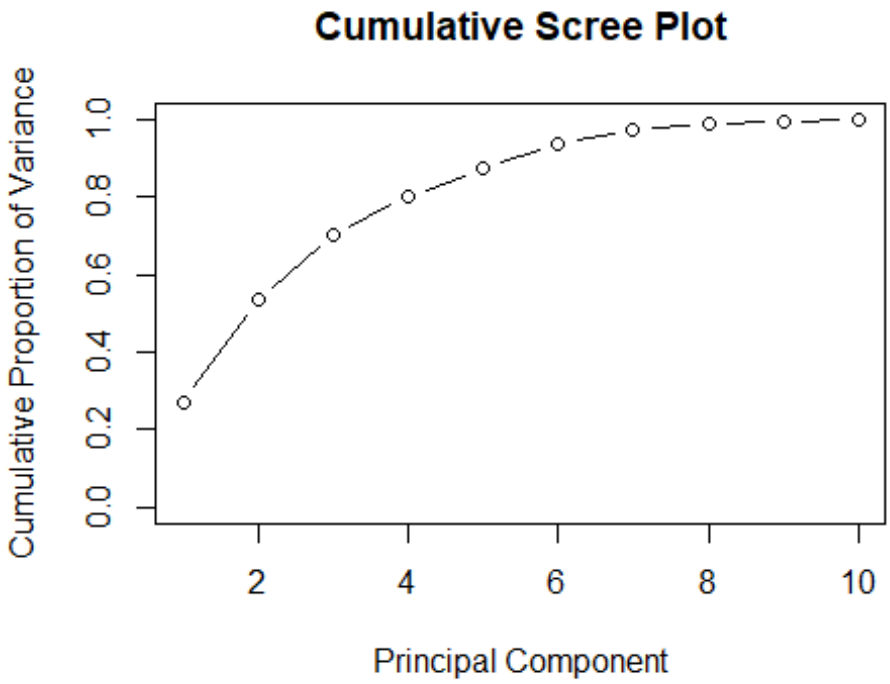
Scree Plot

The scree plot shows the value of q that matches the value of m when the curve falls flat.

```
plot(PVE, xlab="Principal Component", ylab="Proportion of Variance",
main="Scree Plot", ylim=c(0,1), type='b')
```




```
plot(cumsum(PVE), xlab="Principal Component", main="Cumulative Scree Plot",
ylab="Cumulative Proportion of Variance", ylim=c(0,1),type='b')
```



In the "Proportion of variance Explained" plot, the elbow point in not so clear, However, according to this method, it seems reasonable to retain the first six principal components.

Kaiser's rule

The Kaiser's rule suggests to retain as many principal components the ones with the variance greater than 1:

```
pc.out$sdev^2

##  [1] 2.71259568 2.64498398 1.66688369 1.00182221 0.69896455 0.63792379
##  [7] 0.37697179 0.14297211 0.06673271 0.05014949
```

In this case the rule suggests us to retain the first 4 PCs.

##Cluster Analysis:

The purpose of the clustering is to locate homogeneous subgroups in the liver dataset and to accomplish this analysis, various methods can be useful. The analysis composed of many different steps. In the first step, a certain sort of distance is computed among pairs and the distance matrix is established. This is because in this kind of analysis, the concept of dissimilarity is important, since the unit most "similar" will be put into the same cluster, while there must be a large dissimilarity between the other clusters.

First of all we start to create the symmetric matrix, each value represents the distance between units. The values on the diagonal represent the distance between units and

themselves which is zero. To create the distance matrix by default we use an Euclidean Distance:

##Euclidean and Manhattan Distance:

```
df_scaled<-scale(cereal[2:11])
row.names(df_scaled)=cereal$name
dist.eucl<-dist(df_scaled,method = "euclidean")
dist.man<-dist(df_scaled,method = "manhattan")
(round(as.matrix(dist.eucl)[1:5,1:5],2))
```

```
##                           100% Bran 100% Natural Bran All-Bran
## 100% Bran                      0.00              6.52     1.78
## 100% Natural Bran              6.52              0.00     6.99
## All-Bran                       1.78              6.99     0.00
## All-Bran with Extra Fiber      2.77              8.83     3.14
## Almond Delight                 6.61              4.68     6.61
##                           All-Bran with Extra Fiber Almond Delight
## 100% Bran                                      2.77           6.61
## 100% Natural Bran                              8.83           4.68
## All-Bran                                       3.14           6.61
## All-Bran with Extra Fiber                      0.00           8.59
## Almond Delight                                 8.59           0.00
```

```
(round(as.matrix(dist.man)[1:5,1:5],2))
```

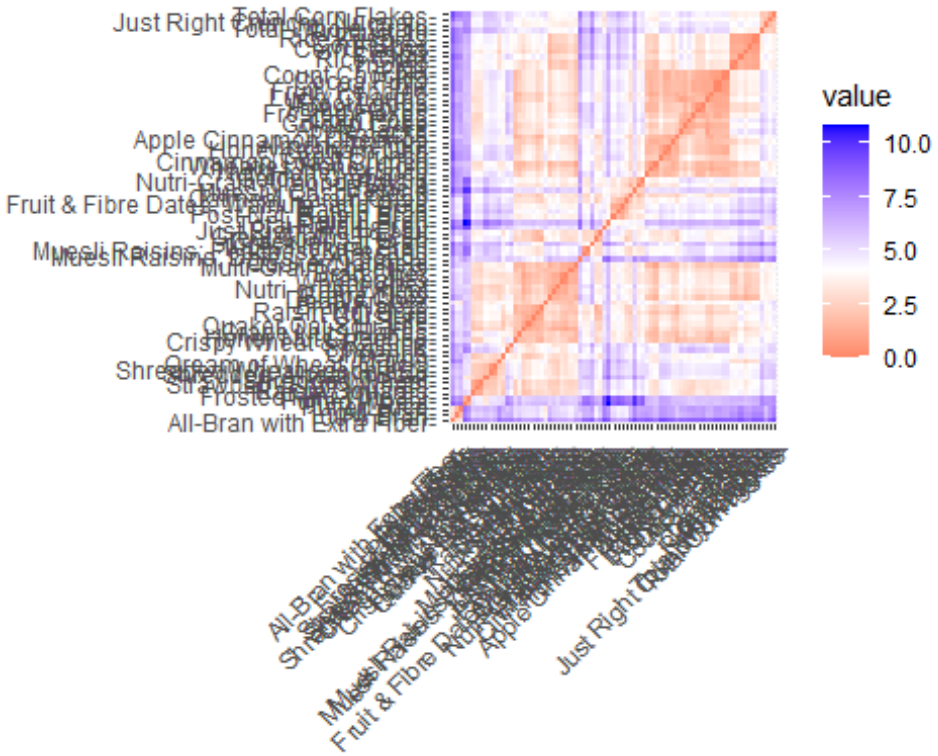
```
##                           100% Bran 100% Natural Bran All-Bran
## 100% Bran                      0.00             16.49     3.22
## 100% Natural Bran             16.49              0.00    17.94
## All-Bran                       3.22             17.94     0.00
## All-Bran with Extra Fiber      6.57             21.65     7.05
## Almond Delight                15.98             11.46    15.76
##                           All-Bran with Extra Fiber Almond Delight
## 100% Bran                                      6.57          15.98
## 100% Natural Bran                             21.65          11.46
## All-Bran                                       7.05          15.76
## All-Bran with Extra Fiber                      0.00          20.91
## Almond Delight                                20.91           0.00
```

As we can see I used the Manhattan distance that we can analyze the incidence of outliers. In this case values seems to maintain the same distance, probably because we are considering a subset.

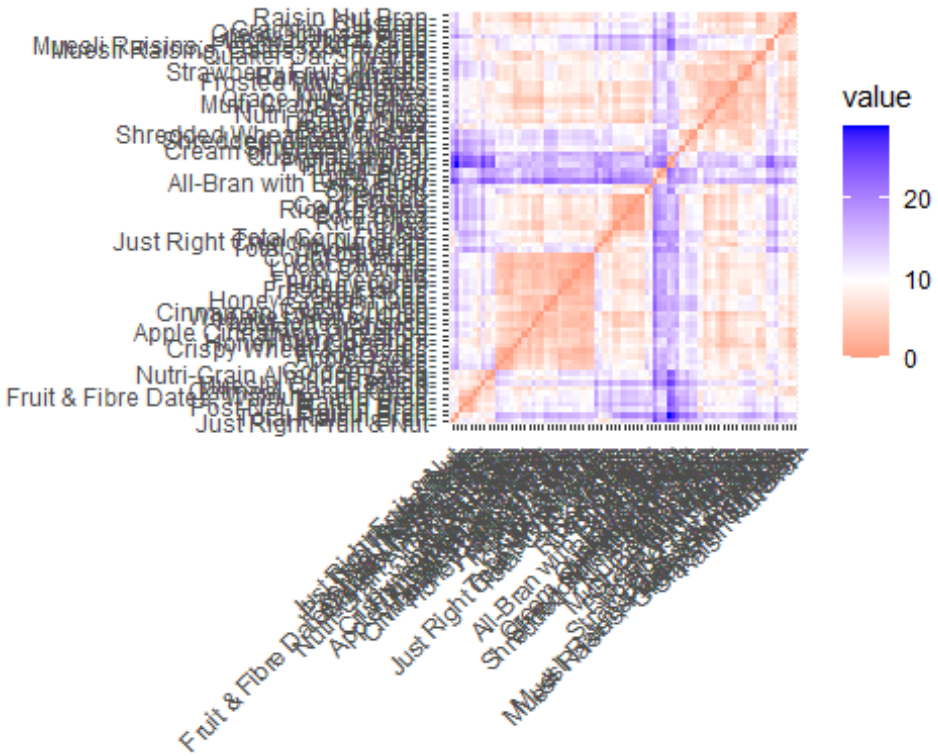
##Visualization

There is a simple solution for visualizing the distance matrix as we can see below, the color level is proportional to the value of the dissimilarity between observations. Red indicates high similarity (i.e. low dissimilarity) while blue indicates low similarity:

```{r} # install.packages("factoextra") #
library(factoextra)

## Warning: package 'factoextra' was built under R version 4.1.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.1.3

##
## Attaching package: 'ggplot2'

## The following objects are masked from 'package:psych':
##
##     %+%, alpha

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

library(ggplot2)
fviz_dist(dist.eucl)
```




```
fviz_dist(dist.man)
```



##Assessing Clustering Tendency

Before applying any clustering method on our data, It's important to evaluate whether the data set contains meaningful cluster or not. In cluster analysis methods return clusters even if the data do not contain clusters.

We create a random data generated from the cereal data set:

```
df_scaled<-scale(cereal[2:11])
```

Random data generated from the Cereal

```
random_df<-apply(df_scaled,2,function(x){runif(length(x),min(x),max(x))})
random_df<-as.data.frame(random_df)
scaled_random_df=scale(random_df)
scaled_random_df
```

```
##          calories     protein         fat       sodium       fiber
carbo
##  [1,] -0.23053046  1.14528156 -0.28583626 -1.491763571 -0.74006858
1.292564593
##  [2,]  0.39545945  0.99899948 -1.19306424  0.514308281  1.74497884 -
0.409628439
##  [3,] -0.40757612 -0.31995646  0.02779019  1.724590194  1.43894277
0.119951561
##  [4,]  0.64858541 -1.52827104 -0.68276581 -1.583596608  1.51836615 -
1.727530260
##  [5,] -0.99274599  0.89555435  0.99089804  0.174262294 -0.89772711 -
```

```
0.813696054
##  [6,] -1.03145439 -0.22524333  0.92235186  0.895481239 -0.31530583
0.223933342
##  [7,]  0.85355171 -1.47801811  0.88211953 -0.351470049  0.07142224 -
1.187674236
##  [8,] -1.02589951  1.07415408 -0.33688952 -1.409033072 -0.30467258
1.438025288
##  [9,]  1.63201889  0.89725531  0.99159979  1.579921052  1.65171518
0.229558166
## [10,]  1.15498193  1.01417624 -1.28146257  0.503680837  0.04294978 -
1.394765396
## [11,] -0.43025532 -1.55673211  0.33469949  1.660886164  1.40164460
1.129676749
## [12,] -0.63602204 -1.27816909 -1.04307709 -1.116776587 -0.95616103
0.006084173
## [13,]  0.66346374  0.70526909 -1.63819879 -1.511087845 -0.91951463
0.898416932
## [14,] -0.68641941 -1.32065082 -1.18076537  0.157925448 -0.76247255
0.049575190
## [15,] -0.20603213 -0.95773871  0.38653349 -0.483696683 -1.29817382 -
0.873078513
## [16,] -1.07874303  0.29393345 -0.20761139  0.913936590 -1.51812381 -
0.975644266
## [17,]  0.23611131  1.16356506  1.27031383  0.345964812 -1.41053947
0.017157925
## [18,] -0.08706952  0.81083075 -0.38129122 -0.279406607  0.09006008
0.193646255
## [19,] -0.29630709 -0.78331090 -0.33853618 -1.795199732 -0.05371722
0.249226171
## [20,]  0.28782871  1.35053926 -1.55801023 -0.676894951  0.92292607
0.105779171
## [21,] -1.25380374  0.13693647  1.29986861  1.753185192  0.33407909 -
1.869588002
## [22,]  1.30005796 -1.44922347 -1.45395738 -0.368018232  0.50048387 -
0.173384304
## [23,] -0.57177362 -0.13524379  0.98108030  1.046005779 -0.90081397
0.816999451
## [24,]  1.70888177 -1.29088489 -0.36227633 -0.277992111 -0.09942096 -
0.100589736
## [25,]  0.19320006 -1.53534646 -1.60960295  0.257493380 -0.82520581
0.627068433
## [26,] -0.80897529 -1.17380041  1.08907299 -0.611166399 -1.50060341 -
1.251652551
## [27,]  0.37357396 -1.35083596 -1.61218009  0.322012250  1.47673144 -
1.866498924
## [28,]  1.05255423  0.01122655 -0.05542449 -1.182225033 -0.57389329
0.415575070
## [29,]  0.66905613 -0.17593851 -1.05879390  0.254950105  0.47392816
0.843767135
## [30,]  1.68870519  0.33299533  0.17422534  0.476195579 -0.41102271
```

```
1.367010297
## [31,] -1.28707239  1.33170902  1.48825380 -0.505888737 -0.53945620 -
1.017667462
## [32,] -0.59580646  0.83008372 -0.10480504  1.461736552 -0.38518805
1.590888694
## [33,]  1.68917848  0.53634216 -0.64508582 -0.359110275  1.60131070
0.692282238
## [34,] -0.02927355 -1.54418105 -1.31678924  0.998044899 -0.45605639 -
0.836360864
## [35,] -1.36401709 -0.46537282  1.59236816  1.627231551 -1.05409044
1.518368362
## [36,] -0.81036000  1.42722851  0.84650105  1.729594953 -0.25841799
0.588341700
## [37,]  1.68758257 -0.84162790  1.40993073  0.390842245 -0.12995932 -
0.216384652
## [38,]  0.33535843 -0.81899994 -1.03957032 -1.388592494 -0.42363974 -
0.333340087
## [39,] -0.30305814  1.34647074 -1.40749710  1.341094833 -1.49167380
0.900913904
## [40,] -0.45045884  1.23222290  0.08697092  0.845315936 -0.71004704
0.296685231
## [41,] -1.07075627 -1.12311812 -0.88104326  1.551885492  0.20075785 -
0.060939353
## [42,]  0.14315875  0.05555158  1.01647883  0.072991859 -0.44088712 -
0.156424950
## [43,] -1.08201278  1.24459279  0.56187540 -0.235295833  0.91100656
1.125033470
## [44,] -1.24152923  0.22099295  0.46177271 -1.479547866  1.46360379 -
1.586153375
## [45,]  0.89740512  1.22157775 -0.72650574  0.061251753 -1.17426537 -
0.641040738
## [46,]  0.66683384  1.38606502  0.63133694  0.723838076  0.76577330 -
1.851278005
## [47,] -1.42555941 -0.56429620  0.53649010  0.224179679  1.19973489
0.004269981
## [48,] -0.37955101 -0.15598220 -0.79426921 -0.535080935  1.09885174
1.212158655
## [49,]  0.07101110  1.14277611  1.02725069 -0.987955281  0.37729838 -
0.287138308
## [50,]  0.92715779 -1.40626525  1.14361447 -0.933597472  1.92592006
0.928743013
## [51,] -0.67309337 -0.47256136  1.31398687 -1.682369708 -0.38115408 -
1.056020826
## [52,] -1.39479124  1.13972686 -0.45893940 -0.835624157  1.26911251 -
1.093744407
## [53,] -0.17907933 -0.13900537  1.28367916 -0.088972870 -0.52563295
1.549143210
## [54,]  1.68653846  0.09853388  0.12443757 -0.855107678 -1.33190193
0.887084859
## [55,] -0.87634272  0.01781901  0.15588447  1.167171768  2.03958323 -
```

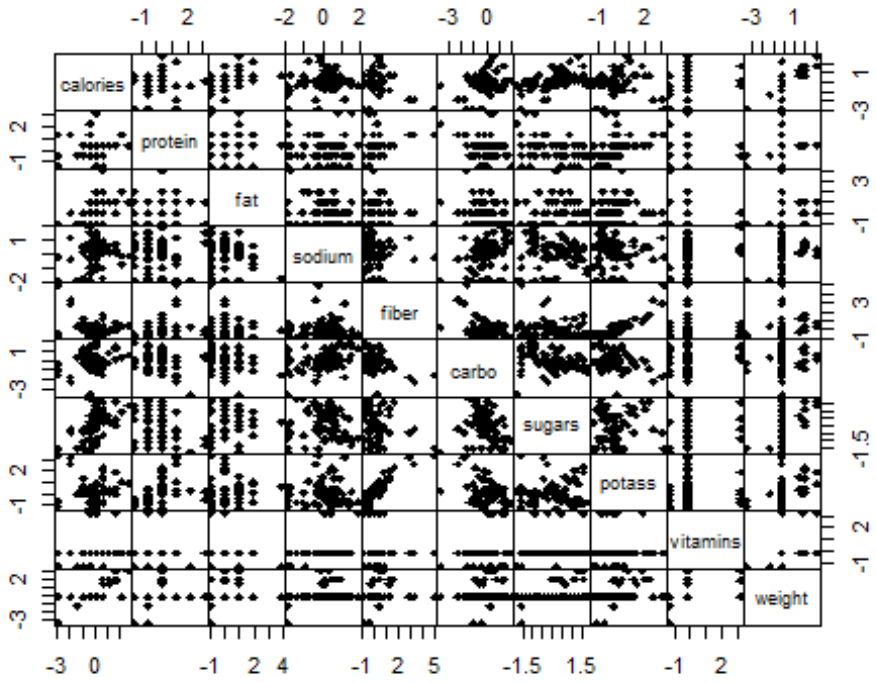
```
0.975973560
## [56,] -1.41910165  1.24670406 -0.39658097  1.031904667 -0.98510022
1.325532521
## [57,]  0.22981275  0.84113157 -0.94461454 -0.056813883 -0.13640885
0.585712955
## [58,]  1.36170382 -0.28405405 -0.84162288  1.039024637 -1.58873486 -
1.584124055
## [59,] -1.24157099 -0.06529623 -0.28019923 -0.870931608 -0.33125616
1.570020061
## [60,]  1.81349945 -1.08150997  0.55339055 -0.004967443  0.13150110
1.511360638
## [61,]  0.81935508 -0.26808667  0.23329159 -0.998713060 -1.20094540 -
1.021359343
## [62,]  1.72222997  0.22949092 -1.06764725  0.673957273  1.57417169 -
0.771253576
## [63,] -1.08915169 -1.01464887  1.39337311  1.228314810  1.37937750 -
0.775400391
## [64,]  0.72013743  0.30231825 -0.80952885 -1.503985828 -1.15046580
0.636272133
## [65,] -0.91898535 -0.97066949  1.15534181  0.182476059 -1.57732947 -
0.935482992
## [66,] -0.76978021 -1.00158918  1.31487456 -1.327095995 -0.45712373 -
0.407291642
## [67,] -0.05425905  1.02521737 -1.69695327 -1.201989747  1.38245126
0.671673566
## [68,]  1.74496584  1.31856309 -1.36363237  0.053556766  0.89863993
1.000897396
## [69,] -1.33278391  0.17512243  0.56361567  1.418225197  0.20651295
1.422256835
## [70,] -0.54170652 -1.43702344  1.08077835  1.155412315 -1.01680466 -
0.986404889
## [71,]  1.53043917  1.38251403  1.40938066 -0.907698055 -0.34275856 -
0.546010072
## [72,] -0.81700748  1.29145403  1.46172337 -0.239643855  0.37077278
0.522495043
## [73,]  0.63394599  0.02544378 -1.25300263  0.294614785  0.06891284
0.883223088
## [74,]  0.25487202 -1.29760984  0.47406624  0.418031445  1.03249325 -
1.687545920
## [75,] -0.47080987  1.36006155 -1.30269451 -0.876593353 -0.18791920 -
0.261780399
## [76,] -1.43157692 -0.57223788  0.05524803 -1.010540126 -0.41005148
1.115974656
## [77,]  1.19988661 -1.17693116  0.88425620 -0.247053012  0.60869102 -
0.830497565
##            sugars      potass    vitamins      weight
##  [1,]  0.52322298  1.17374755 -0.20866279 -1.61509819
##  [2,]  1.19284152  1.40435817  0.09738040 -1.39613723
##  [3,] -0.52354861  0.49336402  0.50775768 -0.81758614
##  [4,]  1.06670291  1.37078675  0.66062850  0.58564473
```

```
##  [5,] -1.29458748 -0.66486124 -0.74537795 -1.29051321
##  [6,] -1.36072486  0.05343032  0.94029585 -0.59118485
##  [7,]  0.34714631  1.04937114 -0.54364781 -0.21868189
##  [8,]  0.06736154 -0.73920717  1.35043263  1.27652370
##  [9,]  0.73895590 -0.39389823 -0.48803138 -1.25252547
## [10,]  1.18805534  0.38954900  1.50740259 -0.69417433
## [11,] -0.03354926  0.38488452 -0.40405225 -0.08590804
## [12,] -1.32896005  1.45033654  0.15755351 -0.77431793
## [13,] -1.22782943  0.90550035  0.47233669 -0.39775872
## [14,]  1.44746953  0.98920255  0.84304688 -1.31948595
## [15,]  1.36759483  1.49338299  1.43679548 -0.11920149
## [16,]  1.04696605  1.22310602  0.67236540  1.79315249
## [17,] -0.84156890 -0.49952559  0.67181545 -1.26468736
## [18,] -1.50644989 -0.60533673 -0.02203492  0.96723437
## [19,]  0.96369472 -1.59486214  0.22949742 -1.62244493
## [20,] -1.04747142 -0.64059286 -0.29651437  1.65575076
## [21,]  0.76679785 -0.12700772  1.62975869 -0.49962609
## [22,] -0.89087177  1.46022619 -0.94802707  0.79538742
## [23,] -0.79035108 -1.42127344 -1.59200822  0.65369432
## [24,]  0.65214147  1.37818335 -0.36356753 -1.02103874
## [25,] -1.29890730  0.12791134 -1.47691597  1.02004085
## [26,] -0.53980288 -0.79682929  0.89834567 -1.37601964
## [27,] -1.43098473 -1.65426081  1.41210172  0.34310441
## [28,] -0.70326723  0.60003403 -0.39234026 -0.37690816
## [29,] -0.03742155 -1.18205226 -1.18796726  1.41699257
## [30,]  0.91415308 -1.48548730  0.89352256  0.60349413
## [31,] -0.07098569 -1.73303710  1.18786168 -0.61234923
## [32,] -0.32175742  0.30445113 -1.71305080 -0.04739694
## [33,]  1.59905198  0.16833268  0.73019974 -0.99901273
## [34,]  0.47567853 -0.81927480  0.07077448 -0.18635603
## [35,]  0.57952375 -1.69561885 -1.21303085  1.34600922
## [36,]  1.48906296  0.79648568  0.66560191 -0.45871027
## [37,] -1.18023574  1.07088087 -0.36233816 -0.02668790
## [38,]  0.97967380  1.02201051 -1.40231922  1.10549694
## [39,] -0.71149735 -0.72769101  0.52389426 -0.55253675
## [40,] -0.20213831  0.27146385  1.44749325 -1.23395301
## [41,]  0.79901908 -0.73266849 -1.45314724  0.39882101
## [42,]  0.88787439  1.20909346  0.69853227 -1.57042852
## [43,] -1.68964848 -0.85616034  0.68783616 -0.99333322
## [44,]  1.20039080 -0.63338882 -1.18760942 -1.36319492
## [45,]  0.12444605 -1.66958336 -1.32992098  1.19400681
## [46,] -1.35792357 -0.22292565 -1.69107515 -1.06266962
## [47,] -0.19840749  0.89171169 -0.25883162  0.96149881
## [48,] -0.89493542  1.11270993  0.96375543  0.94757216
## [49,] -1.03889767  1.13942921 -0.06400546  1.77824614
## [50,] -0.53173687  0.31977452  0.83702953 -0.73250722
## [51,]  0.08760920  1.03712285  1.14714754 -0.06820873
## [52,]  0.70660850 -0.47180329 -0.38582811  1.09362625
## [53,]  1.06305570  0.51762971 -1.61215648 -1.13501397
## [54,]  1.06677792 -0.49828718 -0.12963837 -1.34198379
```

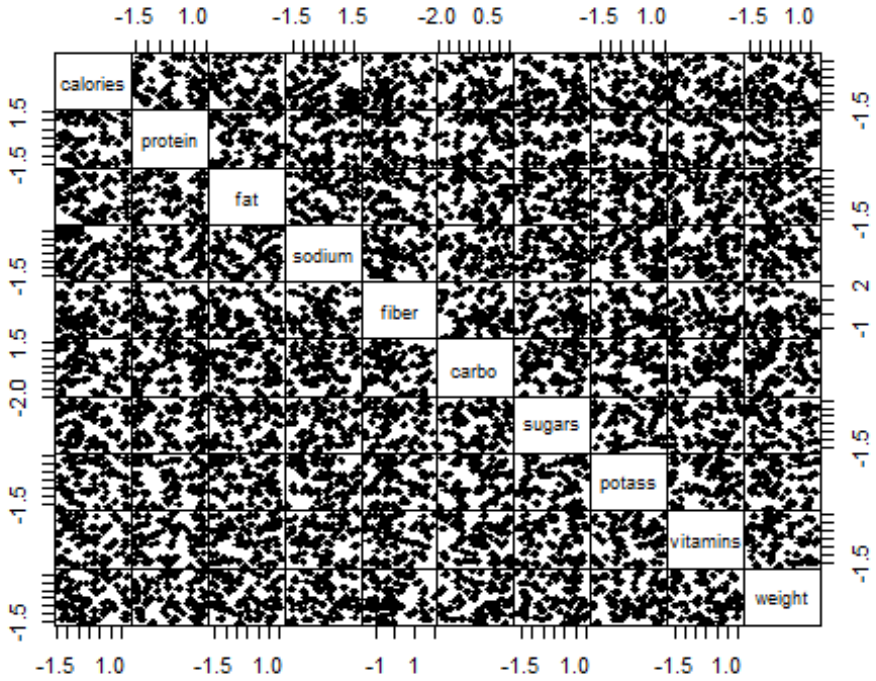
```
## [55,] -1.24735017 -1.07497252  1.55707674  0.08828629
## [56,]  1.35222711 -1.74123161 -1.43941031 -0.46686933
## [57,]  0.60771068  0.14710301  0.14699687  0.53685998
## [58,] -0.53557787  0.27139422  1.20121868  1.53218253
## [59,]  0.81969243 -0.43659166  1.17770579 -0.27605744
## [60,]  0.75833864 -1.77421372 -1.19645929 -0.39983605
## [61,] -1.73139792 -0.73288704 -1.42127444  1.72225856
## [62,]  0.12925022 -0.96129022  0.49863162 -0.34141333
## [63,] -1.28548831 -0.81095093  0.99108438 -0.36210600
## [64,] -1.41721409 -0.32752956  0.70856783 -0.73783757
## [65,] -0.81649236  1.33155392 -1.52148868  0.74657344
## [66,]  1.18085680  1.47594667  0.02301614  1.10257102
## [67,]  1.34378254  1.28015772 -0.98695525 -0.40562447
## [68,]  0.92563937  1.05426439 -1.48615753  1.38720112
## [69,] -1.11538854  0.23536797  0.34062325  1.42998939
## [70,]  1.00836286 -0.20861891  1.51547878 -0.06868545
## [71,] -1.38558516 -0.62043989 -0.88921726  0.39332133
## [72,]  0.97382176 -1.34096850  0.87510572  0.50973621
## [73,]  0.31587274  0.42965020 -1.46242713  1.67273106
## [74,]  0.28004205 -0.26128254 -0.94308220 -0.82393257
## [75,] -1.21003583  0.58358377  0.19981059  0.07480834
## [76,] -0.48515517  0.89063301  0.37548791  0.48710767
## [77,]  1.24667192 -1.35151506 -0.13339591  1.38007937
## attr(,"scaled:center")
##    calories     protein         fat      sodium       fiber       carbo
## -0.44462362  0.95468518  1.56647403  0.02097317  1.58987951 -0.53534333
##      sugars      potass    vitamins      weight
##  0.09506180  1.16290486  1.02426619 -0.37977201
## attr(,"scaled:scale")
## calories  protein      fat   sodium    fiber    carbo   sugars   potass
## 1.695177 1.496531 1.477351 1.056745 1.556451 1.568042 1.071490 1.411771
## vitamins   weight
## 1.323236 1.929123
```

The differences in the distributions can be observed with the graphical pairs representation:

```
pairs(df_scaled,gap=0,pch=18)
```




```
pairs(scaled_random_df,gap=0,pch=18)
```



On the left we have a real representation with standardized data, while on the right we have a representation with Standardized uniform random data from the same data set. We can see that the last representation data do not contain meaningful clusters, while in right representation we can realize the presence at least of 2 clusters.

##Hopkins method For evaluating the clustering tendency we can use the Hopkins statistic method, H that take values in [0,1]. H close to 0 indicates clustered data, H around 0.5 indicates uniformly distributed data (no meaningful clusters).

```
library(clustertend)

## Package `clustertend` is deprecated.  Use package `hopkins` instead.

library(hopkins)

## Warning: package 'hopkins' was built under R version 4.1.3

##
## Attaching package: 'hopkins'

## The following object is masked from 'package:clustertend':
##
##     hopkins

set.seed(123)
hopkins(cereal_sub, m=nrow(cereal_sub)-1)

## [1] 0.9905459

hopkins(scaled_random_df,m=nrow(scaled_random_df)-1)

## [1] 0.3545337
```

As we can see H in real scaled data is equal to 0.37, so this means that the data set is clusterable even if H is not so close to 0 and we could find not well separated groups.

Hierarchical clustering methods

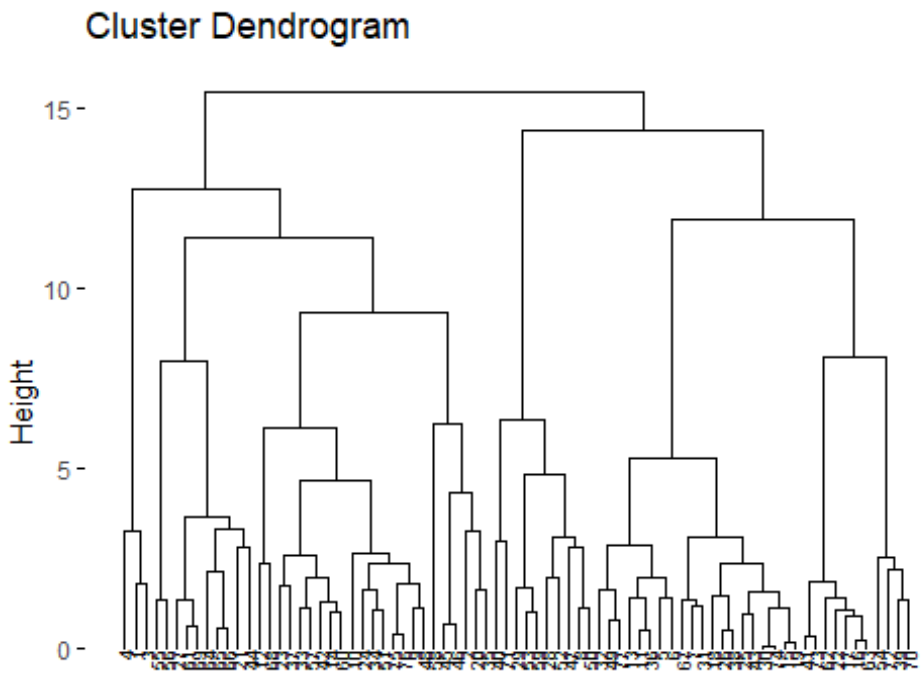
Average linkage method and Euclidean distance means the linkage methods work by calculating the distances or similarities between all objects. Then the closest pair of clusters are combined into a single cluster, reducing the number of clusters remaining. The process is then repeated until there is only a single cluster left.

```
df_scaled_num<-scale(cereal_sub) # scaled with name column
#we can now compute the dissimilarity matrix
rest.dist<-dist(df_scaled_num,method = "euclidean")
#Reformate the result into matrix
as.matrix(rest.dist)[1:6,1:6]

##          1        2        3        4        5        6
## 1 0.000000 6.515095 1.778958 2.770811 6.607588 5.723109
## 2 6.515095 0.000000 6.991328 8.829981 4.678268 4.065401
```

```
## 3 1.778958 6.991328 0.000000 3.135374 6.612588 5.795292
## 4 2.770811 8.829981 3.135374 0.000000 8.590546 7.950381
## 5 6.607588 4.678268 6.612588 8.590546 0.000000 1.401574
## 6 5.723109 4.065401 5.795292 7.950381 1.401574 0.000000
```

Ward's linkage method

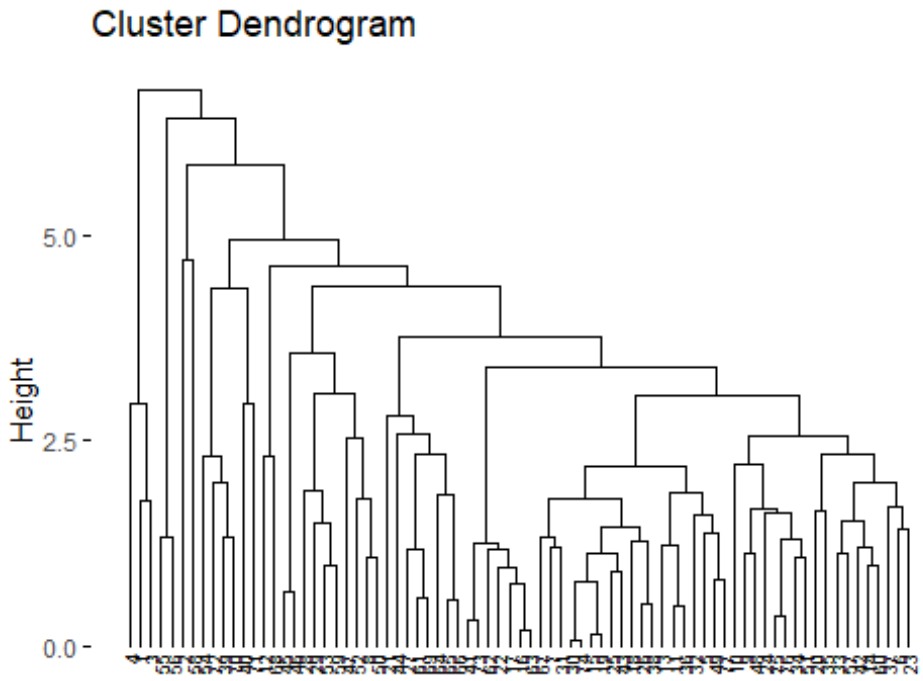
```
res.hc<-hclust(d=rest.dist,method = "ward.D2")
library(factoextra)
fviz_dend(res.hc,cex=0.5)

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



According to the Ward's linkage methods, a cluster is represented by its centroid (as for the centroid linkage method), as we can see I used method= "ward.D2", the only difference from "ward.D" is the distance matrix to be given as an input to hclust(). I detail in the hclust function the input distance in ward.D is considered squared. Looking at the dendogram we can see that the first clusters generated by the algorithm are the linkage between Walnuts and Fruit and Fibre Dates , and at the same height we have the linkage between Dates&almonds and Muesli Raisin. I talked about "height" in fact the dendogram is read by the vertical axis that is represented by the Height. The cophenetic dissimilarity or cophenetic distance of two units is a measure of how similar those two units have to be in order to be grouped into the same cluster. The cophenetic distance between two units is the height of the dendogram where the two branches that include the two units merge into a single branch (height of the fusion). The higher the height of the fusion, the less similar

the units are. After linking the units in a data set into a hierarchical cluster tree, we want to measure how well the cluster tree generated reflects our data, one way to do that is to compute the correlation between the cophenetic distances and the original distances.

```
res.dist<-dist(df_scaled_num,method = "euclidean")
res.hc2<-hclust(d=rest.dist,method = "average")
fviz_dend(res.hc2,cex = 0.5)

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



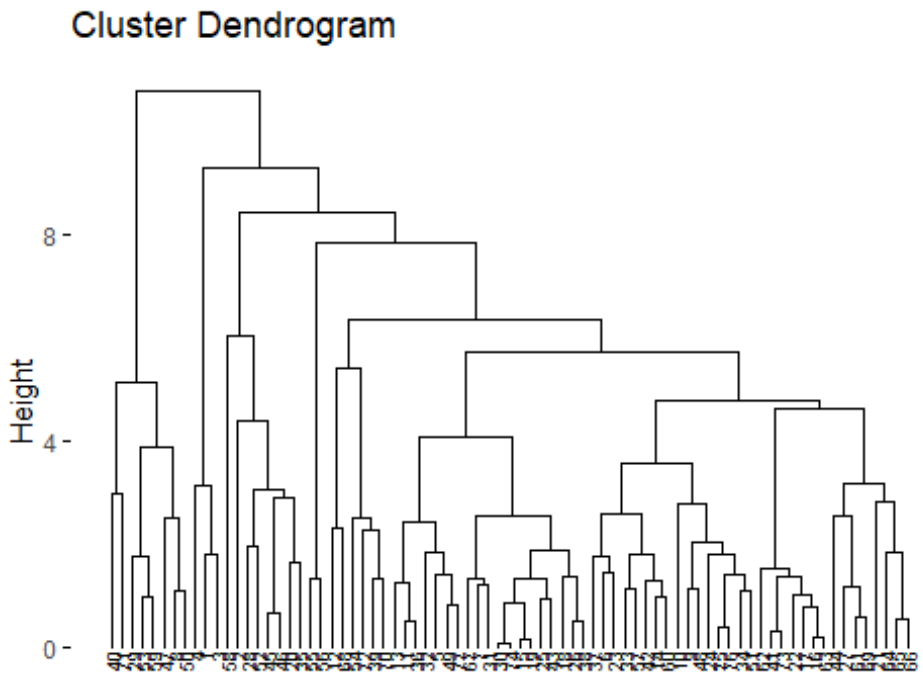

```
res.coph2<-cophenetic(res.hc2)
#correlation between cophenetic and original distances
cor(res.dist,res.coph2)

## [1] 0.8455249
```

As we can see the correlation coefficient shows that using a different linkage method creates a tree that represents the original distances definitely better with a value higher than 0.75.

I used the other two linkage method, respectively single and complete and we can see that using the single method that the coefficient is 0.72 so it is close to an acceptable value that represents the original distance, so remembering that complete linkage gave us a better representation in presence of outliers, we can say that there isn't a significant presence of outliers.

```
res.hc3<-hclust(d = res.dist,method = "complete")
fviz_dend(res.hc3,cex = 0.5)
```

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

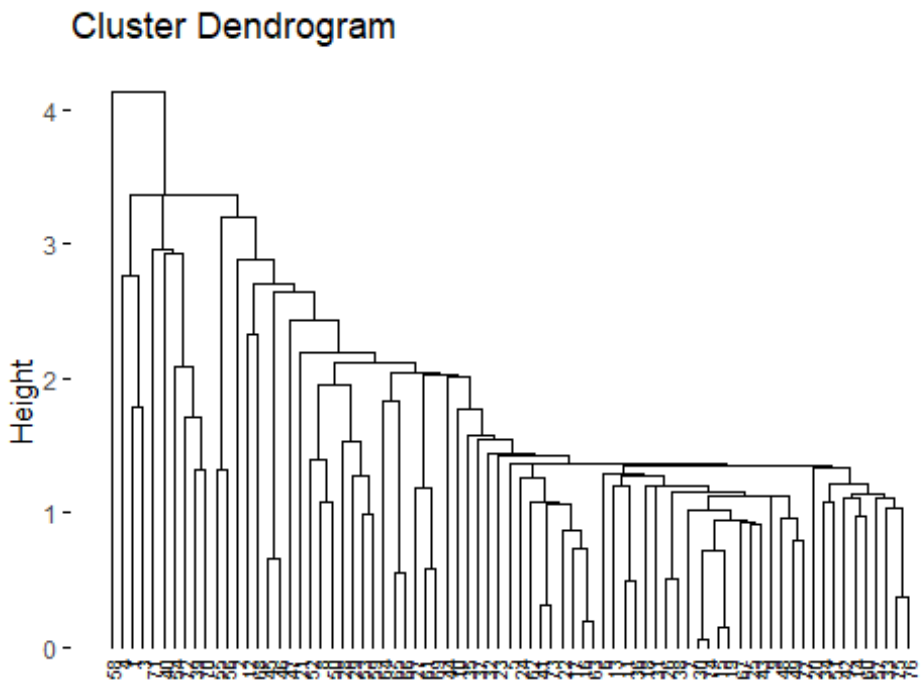



```
res.coph3<-cophenetic(res.hc3)
cor(res.dist,res.coph3)
```

```
## [1] 0.6706783
```

```
res.hc4<-hclust(d = res.dist,method = "single")
fviz_dend(res.hc4,cex = 0.5)
```

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




```
res.coph4<-cophenetic(res.hc4)
cor(res.dist,res.coph4)

## [1] 0.7624329
```

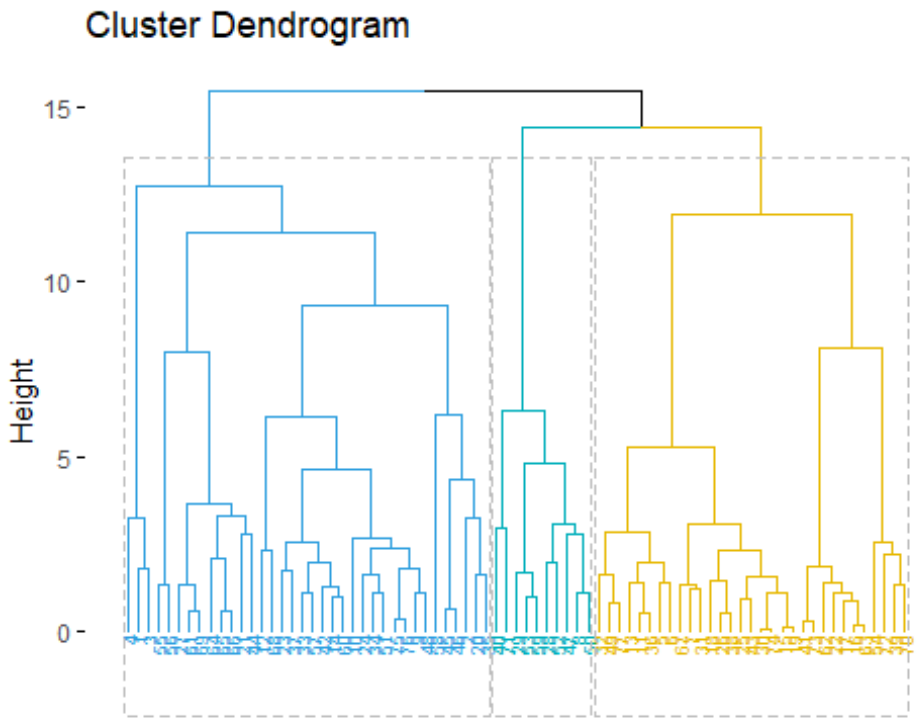
One of the problems with hierarchical clustering is that it does not tell us how many clusters there are, or where to cut the dendogram to form clusters. We can cut the hierarchical tree at a given height in order to partition the data into clusters, specifying the desired number of groups. The function cutree() returns a vector containing the cluster number of each observation.

I cut the dendogram using K=3:

```
# cut tree into 3 groups
grp<-cutree(res.hc,k=3)
head(grp,n=11)

##  [1] 1 1 1 1 2 2 2 3 1 1 2

table(grp)

## grp
##  1  2  3
## 36 31 10
```

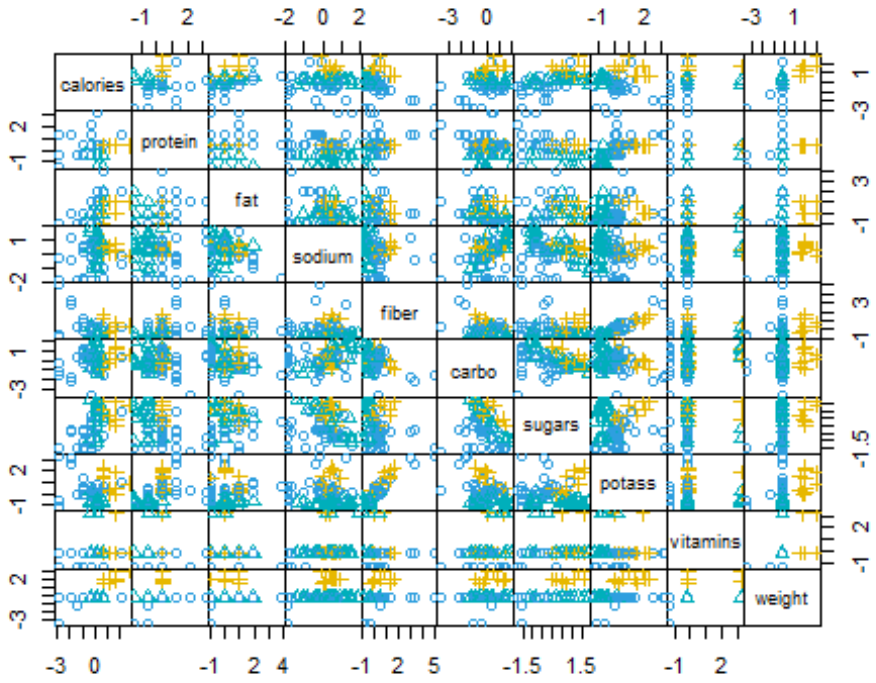
Cutting the dendrogram in 3 groups we can see that in cluster 1 we have 36 units, in cluster 2, 31 units and 10 units in cluster 3. We can visualize the result using the function below:

```
fviz_dend(res.hc,k=3,cex=0.5,k_colors = c("#2E9FDF","#00AFBB","#E7B800"),
          color_labels_by_k = TRUE,rect=TRUE)

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



We can also visualize these clustering result in the original space, via the matrix of pairwise scatterplots:

```
pairs(df_scaled_num,gap =
0,pch=grp,col=c("#2E9FDF","#00AFBB","#E7B800")[grp])
```

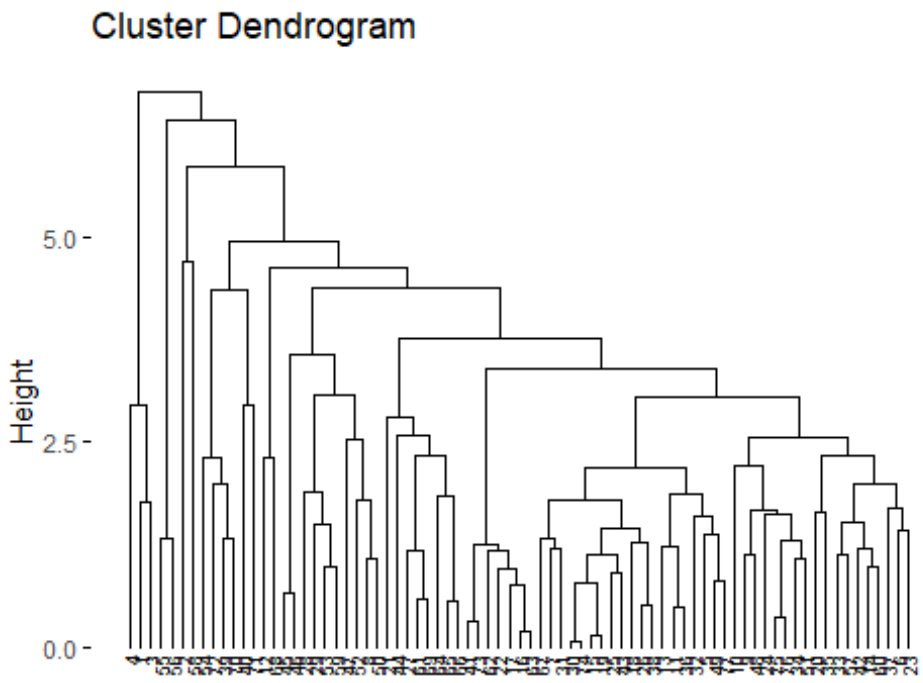


In the scatterplot generated in the original space we can see that in some case is possible to visualize a clustering structure and even different clusters, identified by the different colors used by the hierarchical clustering algorithm, in other there is an overlap and a no defined separation between data.

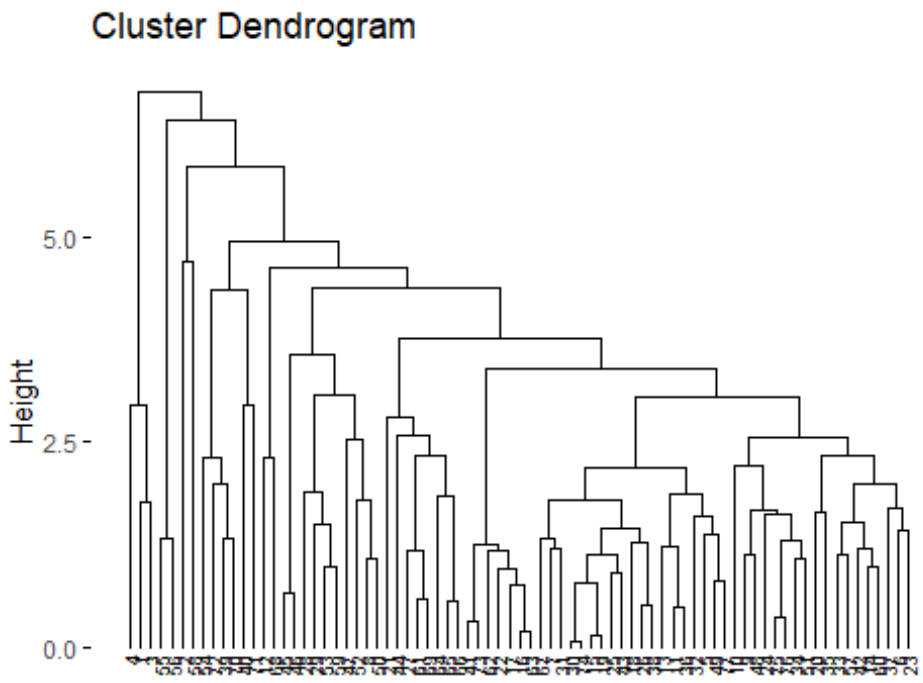
Using the Manhattan distance to evaluate the distance we reduce the incidence of the outliers but as we have seen using the complete method, outliers are not very preponderance in the data. This is confirmed by the correlations below:

```
res.hc2<-hclust(d=res.dist,method = "average")#euclidean distance
fviz_dend(res.hc2,cex = 0.5)
```

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




```
res.coph2<-cophenetic(res.hc2)
#correlation between cophenetic and original distances
cor(res.dist,res.coph2)

## [1] 0.8455249

res.hc5<-hclust(d=res.dist,method = "average") #manhattan distance
fviz_dend(res.hc5,cex = 0.5)

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

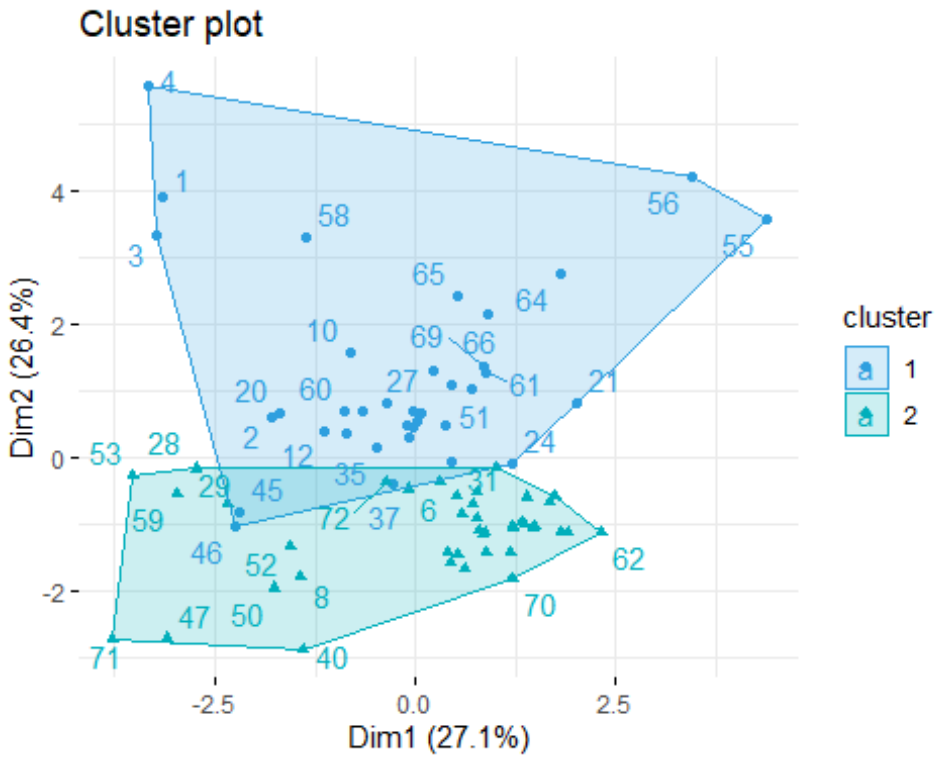



```
res.coph5<-cophenetic(res.hc5)
#correlation between cophenetic and original distances
cor(res.dist,res.coph5)

## [1] 0.8455249
```

In the average linkage method that is that one that represents the data in the best way the value of the correlation using both Euclidean distance and Manhattan are very similar.

We can also see a graphical representation in the original space using the first 2 PC:

```
#cut tree into 2 groups
grp<-cutree(res.hc,k=2) #res.hc2 is a clustering using euclidean dist and avg
method
head(grp,10)

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

fviz_cluster(list(data=df_scaled_num,cluster=grp),palette=c("#2E9FDF","#00AFB
B"),
             ellipse.type = "convex", #concentration ellipse
             repel = TRUE, #void label overploting
             show.clust.cent = FALSE,ggtheme = theme_minimal())

## Warning: ggrepel: 38 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



We can see that in the original space PC1 and PC2 explains about 30% of the total variability, that is not a good result.

##Partitional Clustering methods

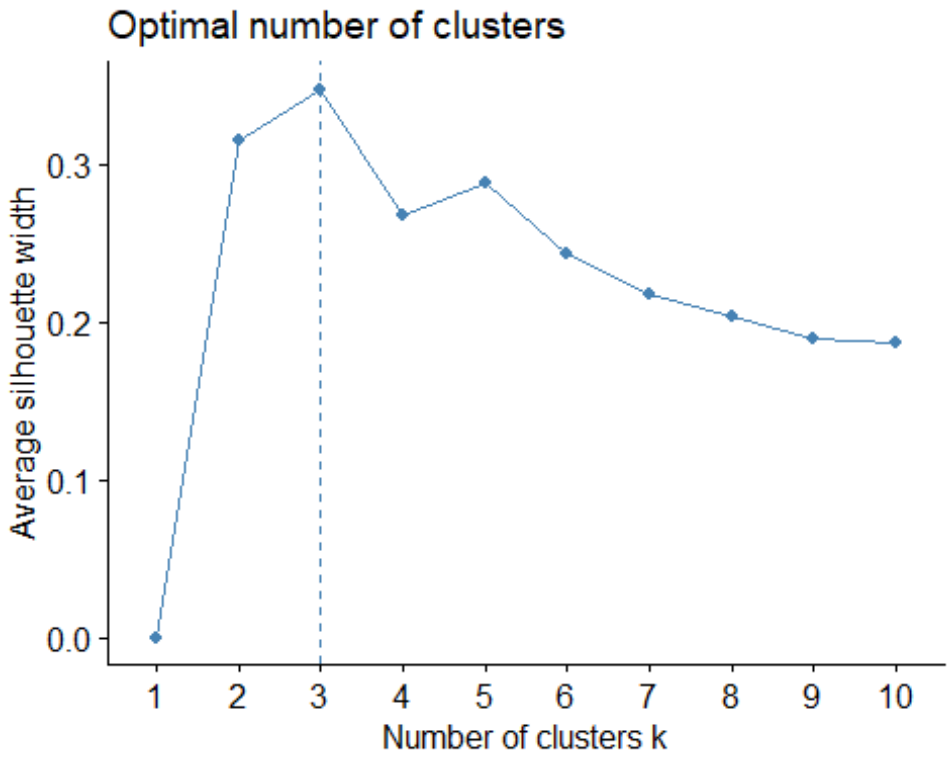
In the partitioning approach the number of clusters K is decided a priori. Once the number of K is specified, the data are split into clusters in such a way that the within-cluster dissimilarity is minimized and the cluster-cohesion is maximized.

K-means clustering is the most commonly used partitioning clustering algorithm. It classifies the n units to the K clusters such that units within the same cluster are as similar as possible, whereas units from different clusters are as dissimilar as possible. Each cluster is represented by the mean of points assigned to the cluster. First of all we said that in the partitional clustering method we have to insert the number of clusters we want a priori but, it's not easy to define the optimal number. So the optimal K depends on the clustering method used. We have two methods for determining the optimal number of cluster for K-means, k-medoids and hierarchical clustering.

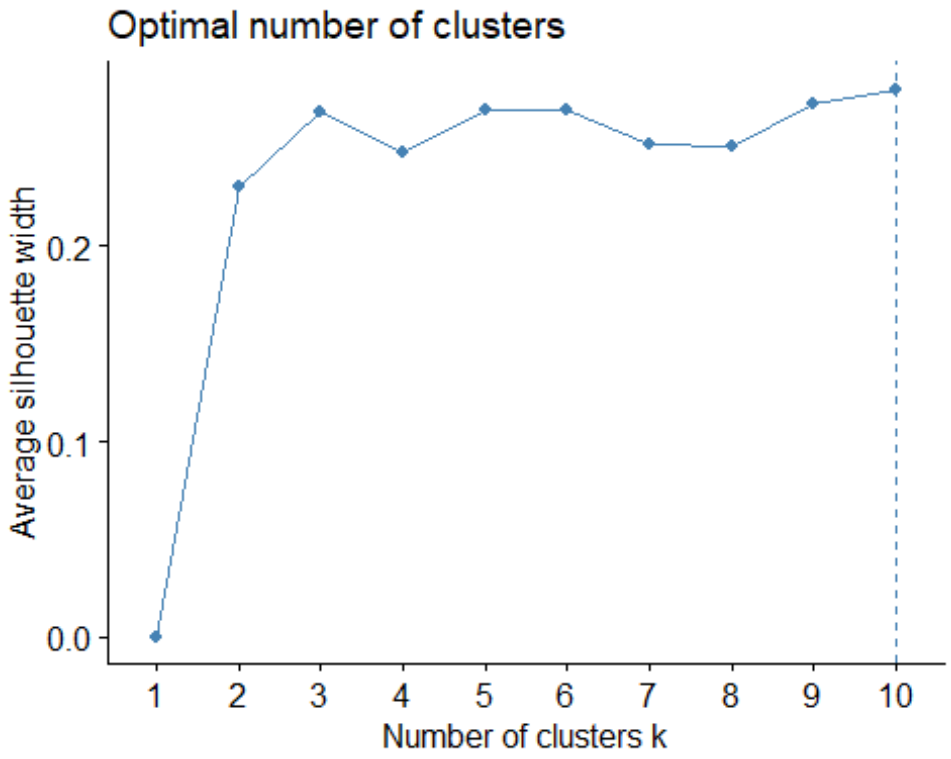
We have direct methods: The Elbow method roughly measures the quality of a clustering by determining how compact clusters are in terms of within-cluster sum of squares, a classical measure of compactness or cohesion. The location of an "elbow" in the plot is generally considered as an indicator of the appropriate number of clusters. The Average silhouette method roughly measures the quality of a clustering by determining how well each unit lies within its cluster. A high average silhouette width indicates a good clustering. Differently from the elbow method, this is a maximization problem.

The Statistical testing methods compare evidence against a null hypothesis of no underlying clustering structure, an example is the Gap statistic that is one of the most popular techniques to determine the optimal number of clusters. The idea is to compare the WSS to its expectations under an appropriate null reference distribution with no obvious clustering, typically a uniform distribution. The estimate of the optimal number of clusters will be the value that maximizes the gap statistic:

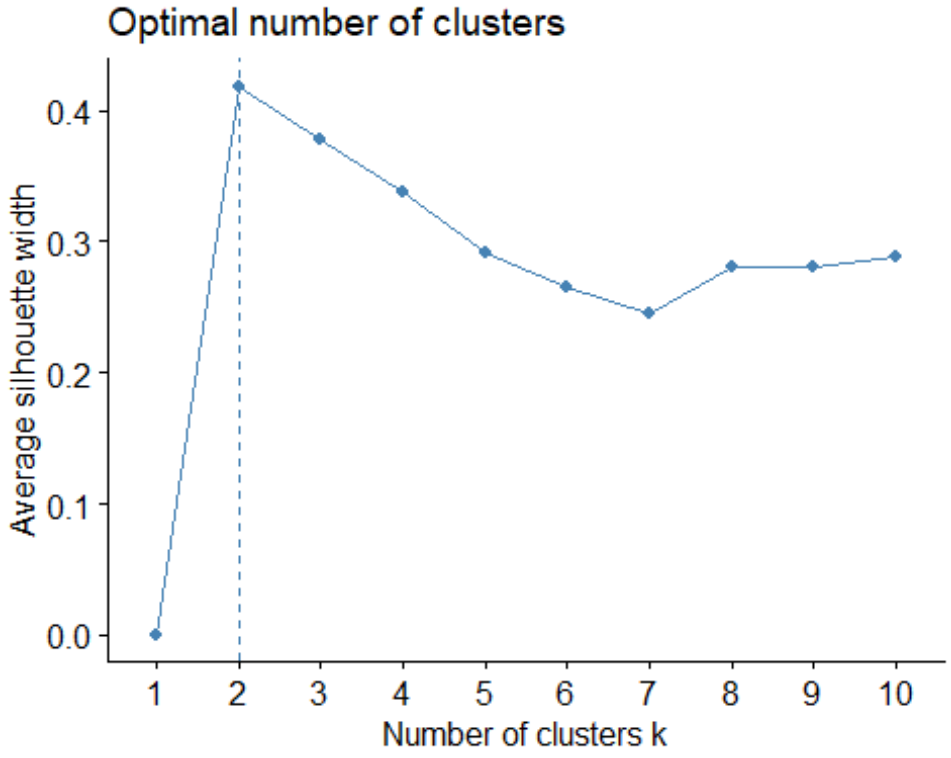
```
fviz_nbclust(df_scaled_num,hcut,method =
"silhouette",hc_metric="euclidean",hc_method="single")
```



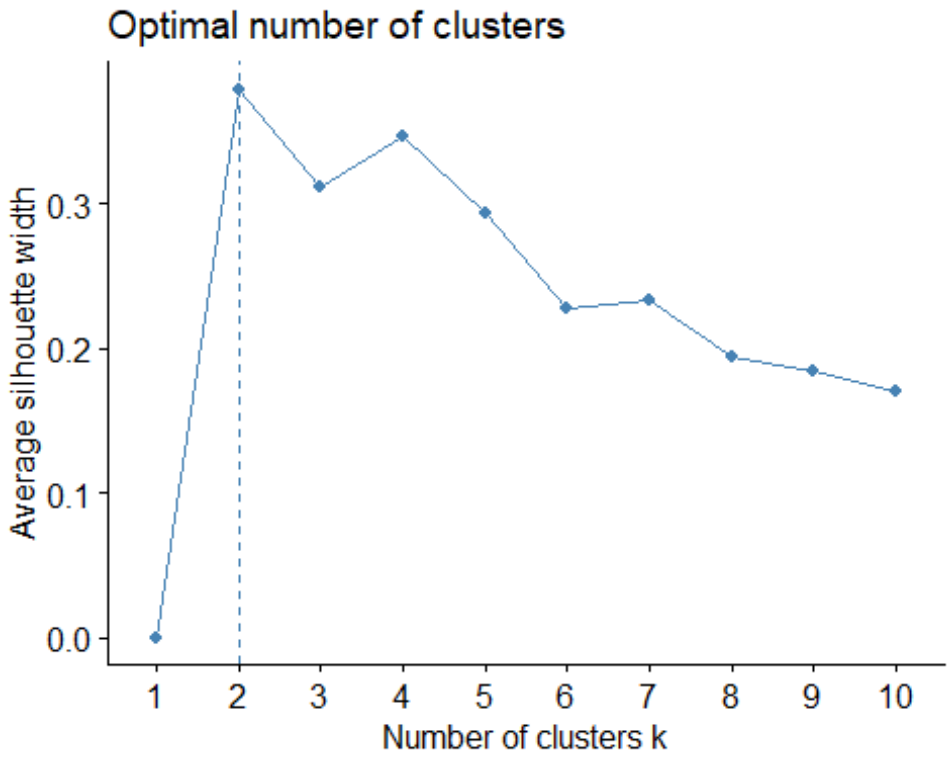

```
fviz_nbclust(df_scaled_num,hcut,method =
"silhouette",hc_metric="euclidean",hc_method="complete")
```




```
fviz_nbclust(df_scaled_num,hcut,method =
"silhouette",hc_metric="euclidean",hc_method="average")
```



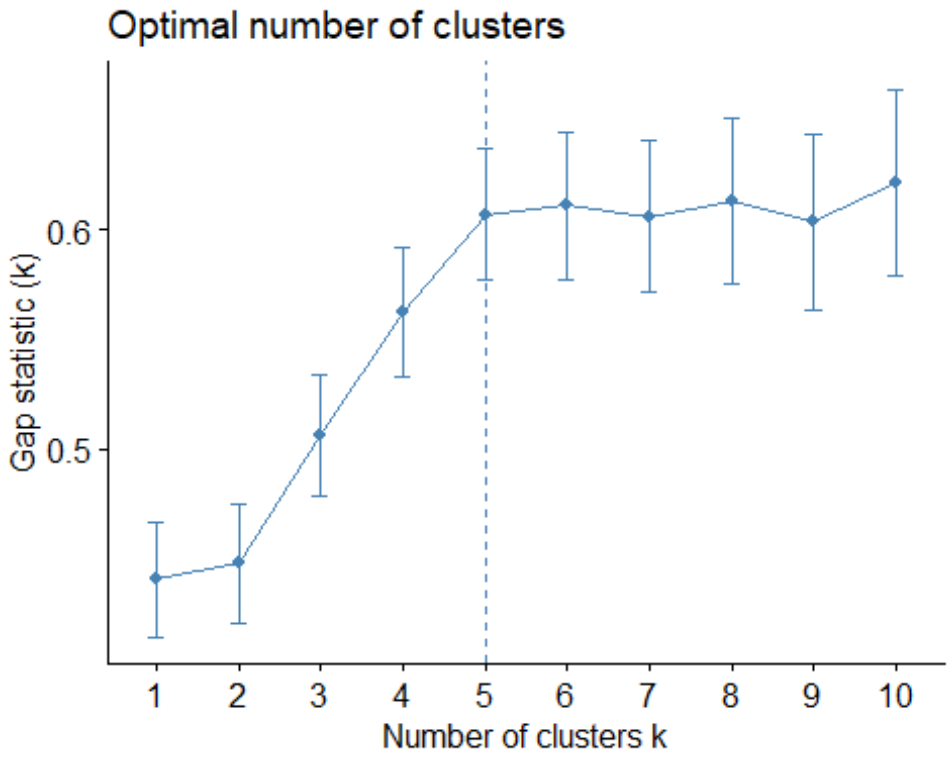
```
fviz_nbclust(df_scaled_num,hcut,method =
"silhouette",hc_metric="euclidean",hc_method="centroid")
```



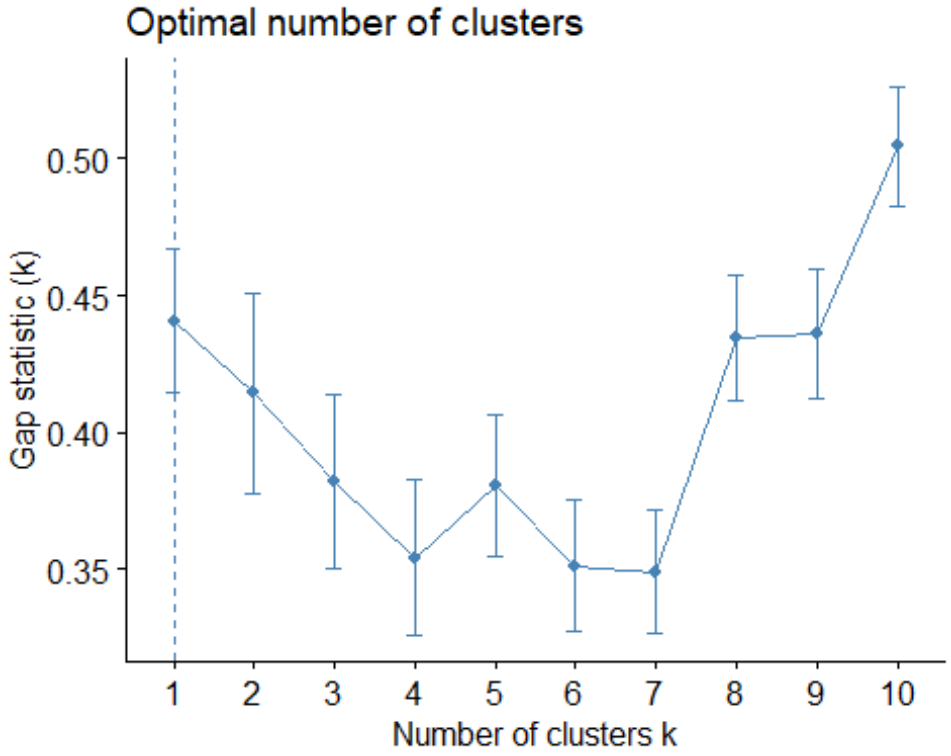
According to the silhouette method the optimal number of clusters in the hierarchical clustering is K=2.

Using the Gap statistic, it suggests to us that the optimal number of clusters is 1:

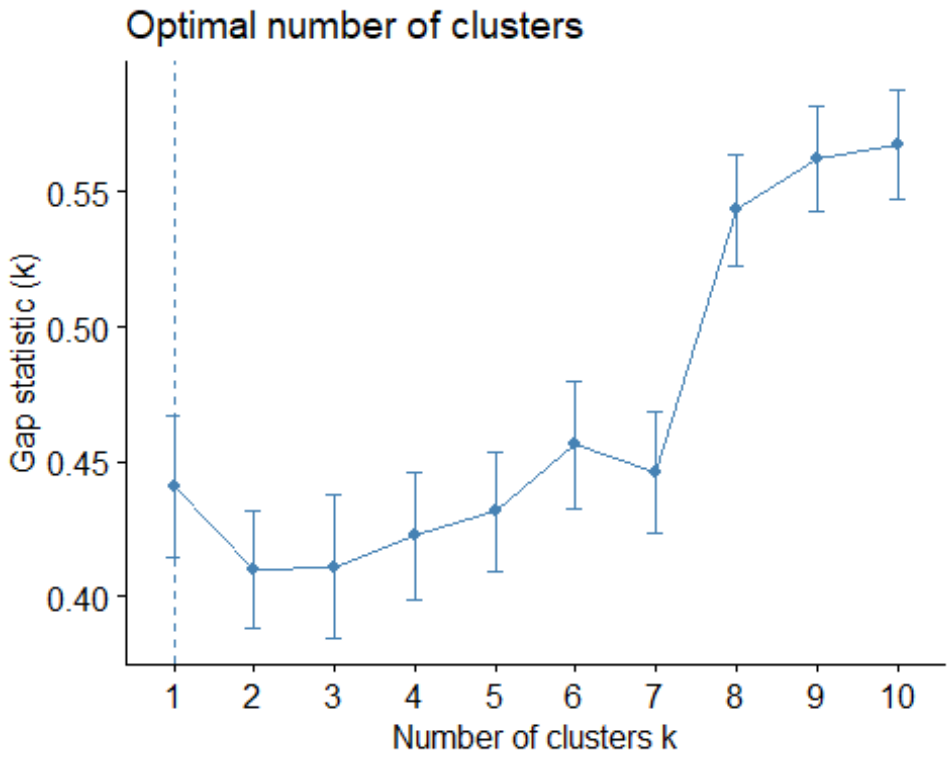
```
#Gap statistic
fviz_nbclust(df_scaled_num,hcut,method =
"gap_stat",hc_metric="euclidean",hc_method="single")
```



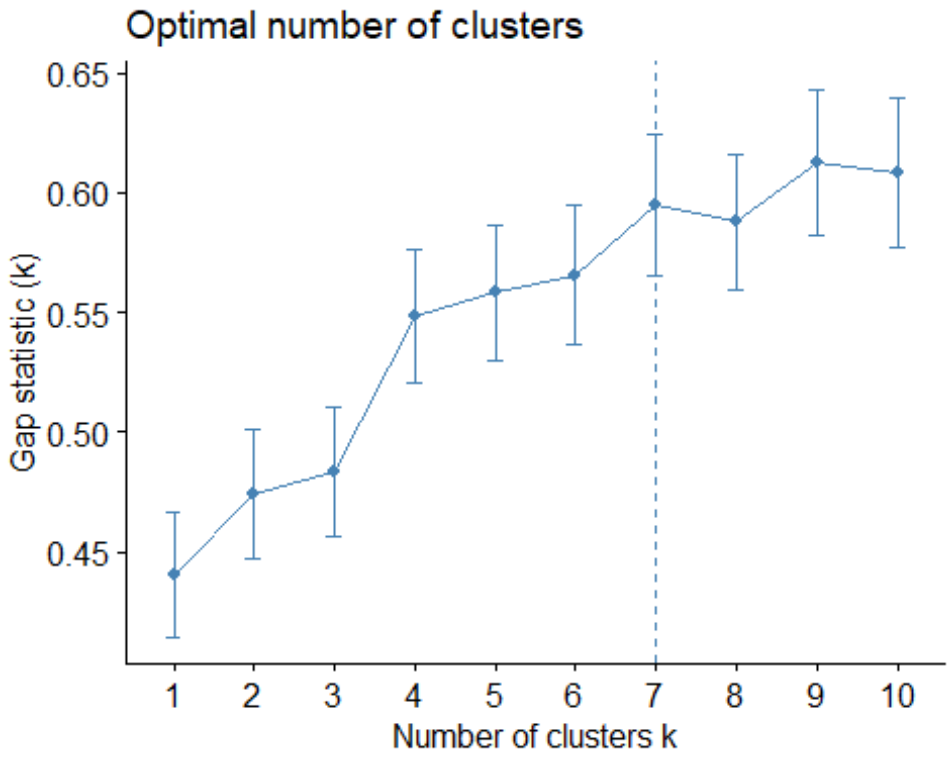

```
fviz_nbclust(df_scaled_num,hcut,method =
"gap_stat",hc_metric="euclidean",hc_method="average")
```



```
fviz_nbclust(df_scaled_num,hcut,method =
"gap_stat",hc_metric="euclidean",hc_method="complete")
```

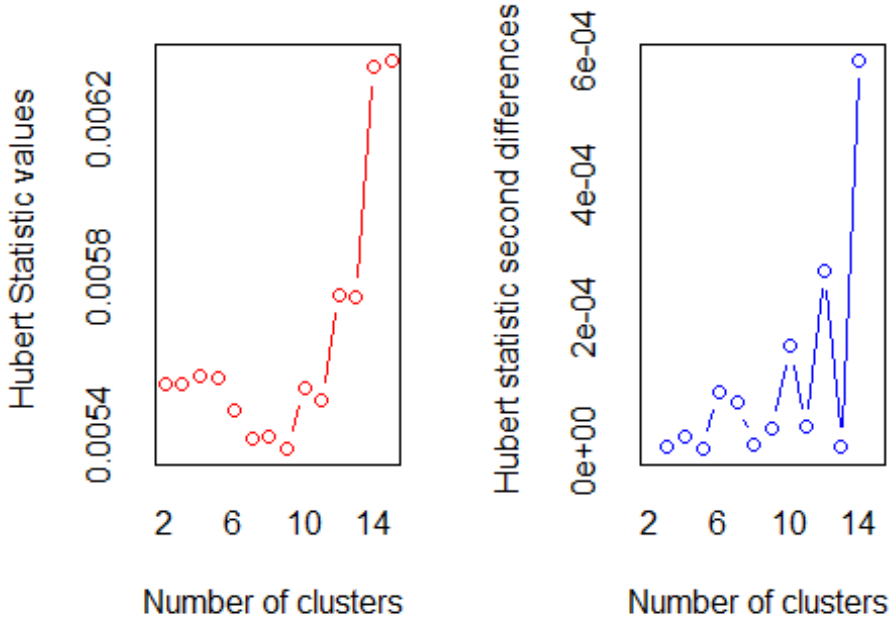



```
fviz_nbclust(df_scaled_num,hcut,method =
"gap_stat",hc_metric="euclidean",hc_method="centroid")
```

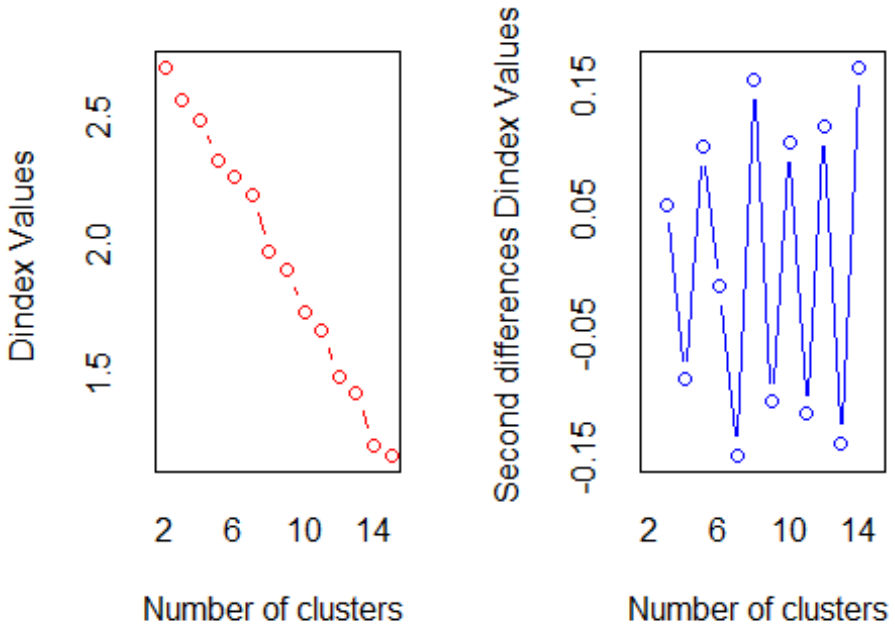


The function NbClust() provides 26 indices for determining the relevant number of clusters and proposes to users the best clustering scheme from the different results obtained by varying all combinations of number of clusters, distance measures, and clustering.

```
library(NbClust)

## Warning: package 'NbClust' was built under R version 4.1.3

NbClust(df_scaled_num,distance = "euclidean", method = "average")

## Warning in pf(beale, pp, df2): NaNs produced

## Warning in pf(beale, pp, df2): NaNs produced
```




```
## *** : 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:
## * 6 proposed 2 as the best number of clusters
## * 3 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 4 proposed 5 as the best number of clusters
## * 1 proposed 7 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 1 proposed 11 as the best number of clusters
## * 5 proposed 14 as the best number of clusters
## * 1 proposed 15 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

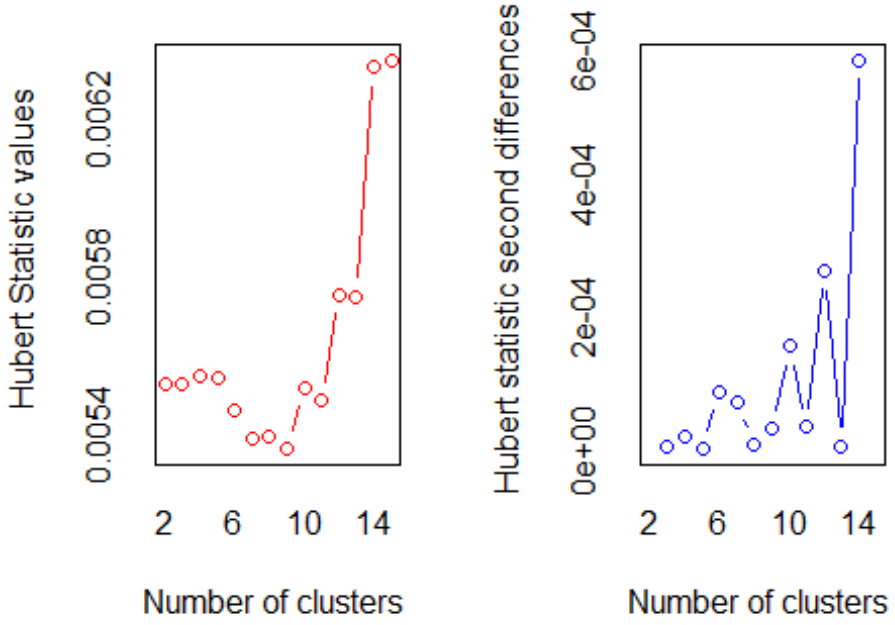
```
## $All.index
##         KL      CH Hartigan     CCC     Scott      Marriot    TrCovW
TraceW
## 2   0.3989 12.0656   8.5556 -3.7816   90.2590 7.673823e+15 5546.6424
654.6789
## 3   1.4034 10.8521   5.9374 -6.3029  133.0295 9.907412e+15 5003.7161
587.6436
## 4   0.2118  9.6620  11.2184 -7.9574  205.2676 6.892889e+15 4046.1080
543.9959
## 5   8.0106 11.0118   1.7230 -6.9691  454.2802 4.243689e+14 3576.9556
471.5323
## 6   0.4081  9.2348   4.5812 -8.8538  539.0888 2.031280e+14 3472.1689
460.5120
## 7   0.1693  8.8296  18.7868 -9.4314  604.0826 1.188737e+14 3136.2661
432.5991
## 8   5.6086 12.1078   4.7336 -5.1027  709.8189 3.932760e+13 2535.3208
341.0636
## 9   0.2786 11.7401  14.3739 -4.7487  795.9535 1.626248e+13 2415.3357
319.1678
## 10  4.4443 14.0293   4.2743 -1.0148  914.1453 4.325914e+12 1718.4261
263.4745
## 11  0.2468 13.6524  15.6146 -0.8279 1091.2510 5.247521e+11 1623.7990
247.6741
## 12  4.0194 16.5131   4.7862  3.1695 1160.3352 2.546144e+11  947.7801
200.2889
## 13  0.2160 16.3943  24.0042  3.5399 1205.9939 1.651519e+11  848.8818
186.5523
## 14 18.8923 22.3016   2.0980  9.8370 1315.5915 4.614257e+10  375.3547
135.6678
## 15  0.6845 21.2061   2.6124  9.2108 1364.8812 2.792705e+10  360.1018
131.2955
##    Friedman  Rubin Cindex     DB Silhouette   Duda Pseudot2   Beale
Ratkowsky
## 2   12.5175 1.1609 0.3611 0.7403     0.4181 0.8965   8.3108  0.7656
0.1971
## 3   18.2903 1.2933 0.4408 0.6769     0.3775 0.9247   5.6978  0.5397
0.2508
## 4   22.3575 1.3971 0.4365 0.8549     0.3375 0.8620  10.8836  1.0608
0.2562
## 5   49.9905 1.6118 0.4595 0.9978     0.2922 4.3984   0.0000  0.0000
0.2634
## 6   61.6272 1.6503 0.4594 0.8198     0.2914 0.9335   4.4139  0.4712
0.2468
## 7   65.3169 1.7568 0.4491 0.8280     0.2707 0.7649  18.4391  2.0328
0.2419
## 8   80.4236 2.2283 0.5058 0.8279     0.3067 0.3315   8.0647 10.8471
0.2573
## 9   88.8223 2.3812 0.5173 0.8135     0.3070 0.7834  13.8270  1.8233
0.2488
## 10 104.4005 2.8845 0.4899 0.8286     0.3147 0.7144   3.1983  2.3899
```

```
0.2532
## 11 171.8164 3.0685 0.4874 0.7950     0.3242 0.7396  14.7895  2.3130
0.2452
## 12 194.4874 3.7945 0.5014 0.7759     0.3306 0.4450   7.4830  7.1891
0.2466
## 13 205.7770 4.0739 0.4988 0.7532     0.3406 0.6113  22.2507  4.1566
0.2399
## 14 229.4415 5.6019 0.4727 0.7828     0.3803 8.1082   0.0000  0.0000
0.2416
## 15 276.1417 5.7885 0.4721 0.7273     0.3977 3.6490  -0.7259 -2.4410
0.2342
##        Ball Ptbiserial   Frey McClain   Dunn Hubert SDindex Dindex   SDbw
## 2  327.3394     0.4574 1.0128  0.0478 0.3113 0.0055  0.9011 2.6944 0.6268
## 3  195.8812     0.5711 1.8261  0.0819 0.3741 0.0055  0.6861 2.5662 0.4222
## 4  135.9990     0.6303 2.7101  0.1200 0.3444 0.0055  0.8301 2.4919 0.5274
## 5   94.3065     0.6535 1.9974  0.2609 0.3873 0.0055  0.9473 2.3309 0.5569
## 6   76.7520     0.6535 1.8565  0.2613 0.3873 0.0055  0.7017 2.2699 0.3133
## 7   61.7999     0.6570 1.1375  0.3177 0.3636 0.0054  0.6976 2.1972 0.3093
## 8   42.6329     0.6736 0.5562  0.6348 0.3614 0.0054  0.7810 1.9767 0.3228
## 9   35.4631     0.6745 1.0143  0.6406 0.3705 0.0054  0.8067 1.9102 0.2681
## 10  26.3474     0.6443 0.8622  1.0143 0.3642 0.0055  0.9023 1.7390 0.2597
## 11  22.5158     0.6425 0.8022  1.0363 0.3642 0.0055  0.8786 1.6718 0.2281
## 12  16.6907     0.6030 0.9664  1.5022 0.2567 0.0058  0.8761 1.4902 0.2099
## 13  14.3502     0.5989 0.4585  1.5421 0.2567 0.0057  0.9274 1.4255 0.1966
## 14   9.6906     0.5285 0.2469  2.7829 0.2348 0.0063  1.0382 1.2227 0.1950
## 15   8.7530     0.5283 0.2464  2.7879 0.2348 0.0063  1.0537 1.1842 0.1582
##
## $All.CriticalValues
##    CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
## 2          0.7769            20.6805       0.6622
## 3          0.7747            20.3619       0.8624
## 4          0.7724            20.0408       0.3906
## 5         -0.0337             0.0000          NaN
## 6          0.7649            19.0610       0.9088
## 7          0.7621            18.7286       0.0281
## 8          0.3763             6.6304       0.0000
## 9          0.7461            17.0155       0.0541
## 10         0.5025             7.9198       0.0155
## 11         0.7295            15.5719       0.0118
## 12         0.4513             7.2945       0.0000
## 13         0.7108            14.2399       0.0000
## 14        -0.0337             0.0000          NaN
## 15         0.1443             5.9302       1.0000
##
## $Best.nc
##                       KL      CH Hartigan    CCC    Scott      Marriot
TrCovW
## Number_clusters 14.0000 14.0000  14.0000 14.000   5.0000 5.000000e+00
4.0000
## Value_Index     18.8923 22.3016  21.9062  9.837 249.0126 6.247279e+15
```

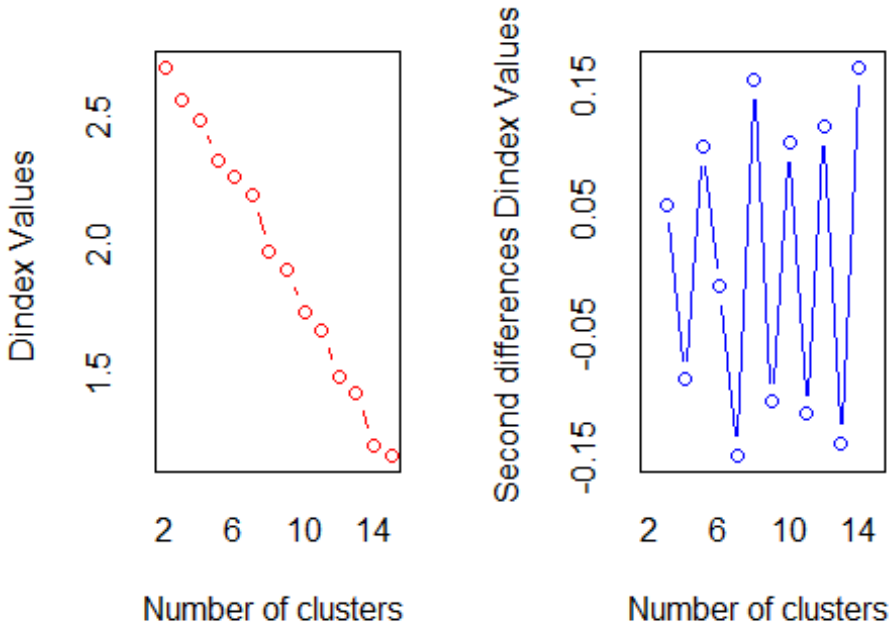
```
957.6081
##                   TraceW Friedman   Rubin Cindex     DB Silhouette   Duda
## Number_clusters  8.0000  11.0000 14.0000 2.0000 3.0000     2.0000 2.0000
## Value_Index     69.6398  67.4158 -1.3414 0.3611 0.6769     0.4181 0.8965
##                  PseudoT2  Beale Ratkowsky     Ball PtBiserial   Frey
McClain
## Number_clusters   2.0000 2.0000    5.0000   3.0000     9.0000 7.0000
2.0000
## Value_Index       8.3108 0.7656    0.2634 131.4582     0.6745 1.1375
0.0478
##                   Dunn Hubert SDindex Dindex    SDbw
## Number_clusters 5.0000      0  3.0000      0 15.0000
## Value_Index     0.3873      0  0.6861      0  0.1582
##
## $Best.partition
##  [1] 1 2 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 2 2 2 2 2 2 2
2 2 2
## [39] 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 2
## [77] 2
```

The same value we will find in the histogram below using in the factoextra package:

```
install.packages("NbClust")

## Warning: package 'NbClust' is in use and will not be installed

library(NbClust)
library(factoextra)
n<-NbClust(df_scaled_num,distance = "euclidean",method = "average")

## Warning in pf(beale, pp, df2): NaNs produced

## Warning in pf(beale, pp, df2): NaNs produced
```

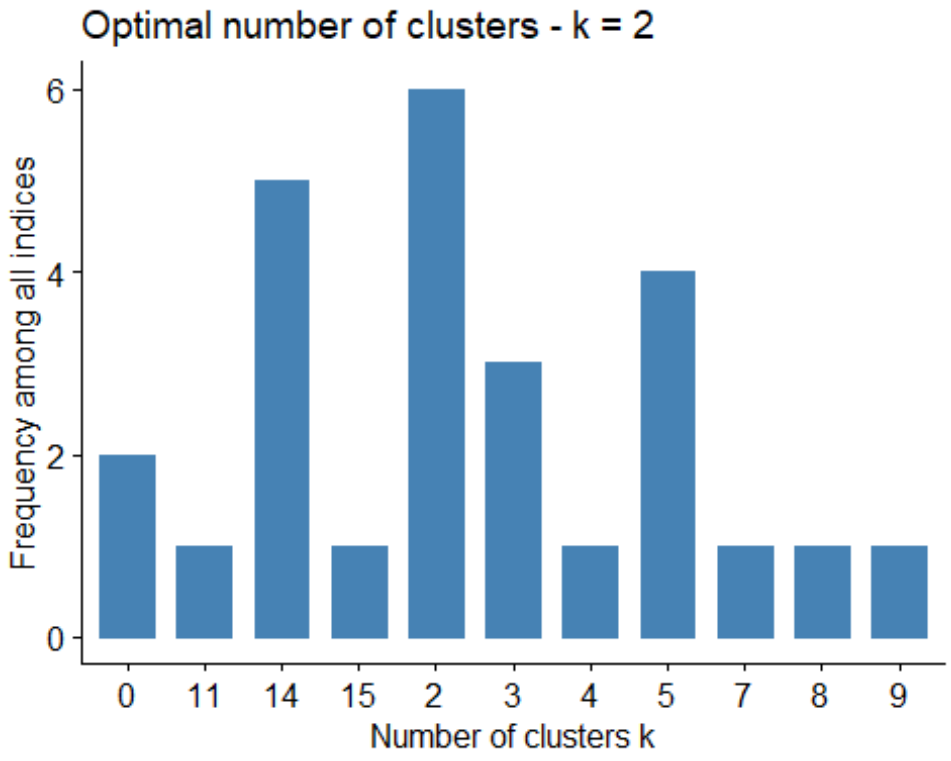



```
## *** : The Hubert index is a graphical method of determining the number of
clusters.
##                 In the plot of Hubert index, we seek a significant knee
that corresponds to a
##                 significant increase of the value of the measure i.e the
significant peak in Hubert
##                 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:
## * 6 proposed 2 as the best number of clusters
## * 3 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 4 proposed 5 as the best number of clusters
## * 1 proposed 7 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 1 proposed 11 as the best number of clusters
## * 5 proposed 14 as the best number of clusters
## * 1 proposed 15 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

```
fviz_nbclust(n)

## Warning in if (class(best_nc) == "numeric") print(best_nc) else if
## (class(best_nc) == : the condition has length > 1 and only the first
element
## will be used

## Warning in if (class(best_nc) == "matrix") .viz_NbClust(x, print.summary,
: the
## condition has length > 1 and only the first element will be used

## Warning in if (class(best_nc) == "numeric") print(best_nc) else if
## (class(best_nc) == : the condition has length > 1 and only the first
element
## will be used

## Warning in if (class(best_nc) == "matrix") {: the condition has length > 1
and
## only the first element will be used

## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 6 proposed  2 as the best number of clusters
## * 3 proposed  3 as the best number of clusters
## * 1 proposed  4 as the best number of clusters
## * 4 proposed  5 as the best number of clusters
## * 1 proposed  7 as the best number of clusters
## * 1 proposed  8 as the best number of clusters
## * 1 proposed  9 as the best number of clusters
## * 1 proposed  11 as the best number of clusters
## * 5 proposed  14 as the best number of clusters
## * 1 proposed  15 as the best number of clusters
##
## Conclusion
## =========================
## * According to the majority rule, the best number of clusters is  2 .
```

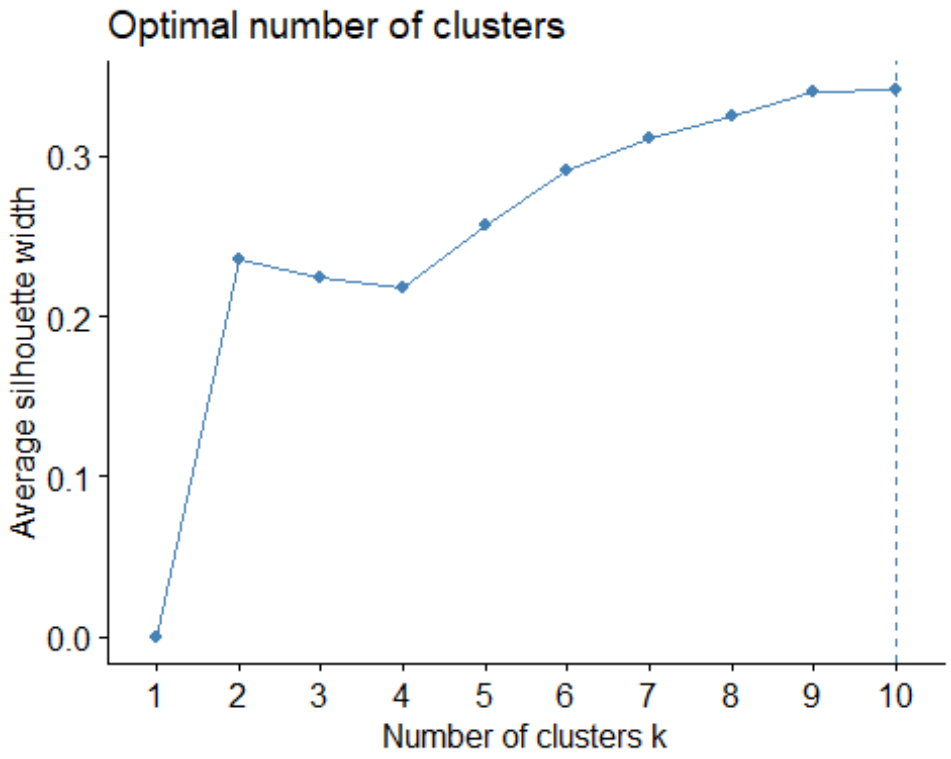


So, now we have decided the optimal K, we apply the partitional clustering methods using the k-means and the k-medoids.

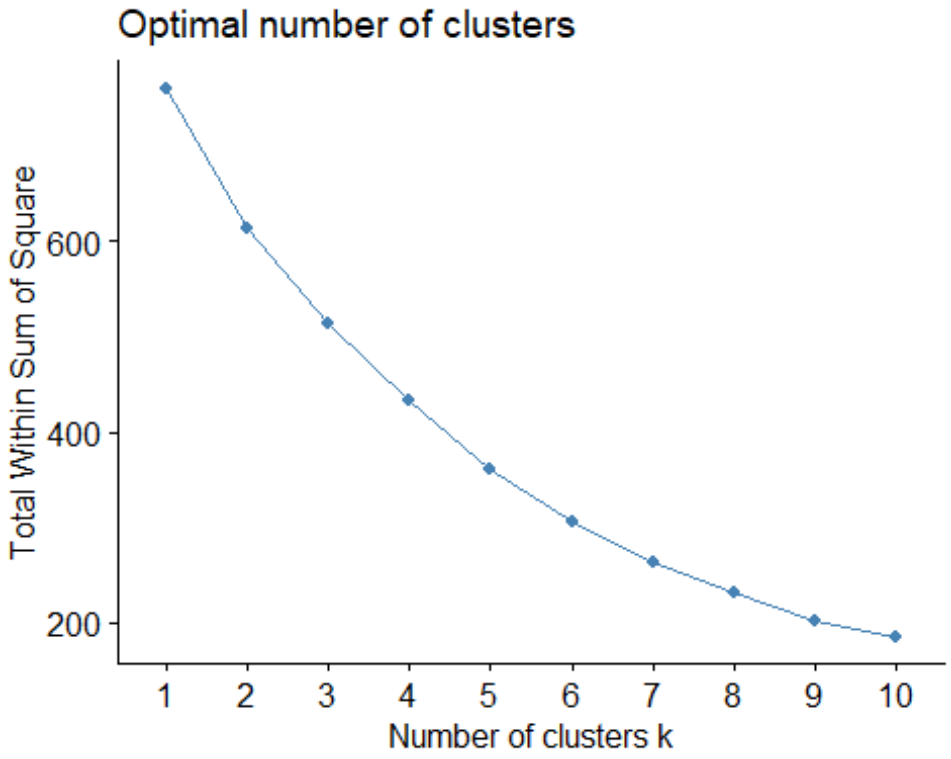
k-means

In the k-means clustering each cluster is represented by the mean of points assigned to the cluster. The algorithm selects randomly k points (as the number of clusters we set in the beginning) from the data set (rows of X) to serve as the initial centers (centroids) for the clusters. Each observation is assigned to the closest centroid using the Euclidean distance between the observation and the cluster mean, then it is updated the new mean of all data points in that cluster. The k-means algorithm minimizes the objective function WSS (within sum of square).

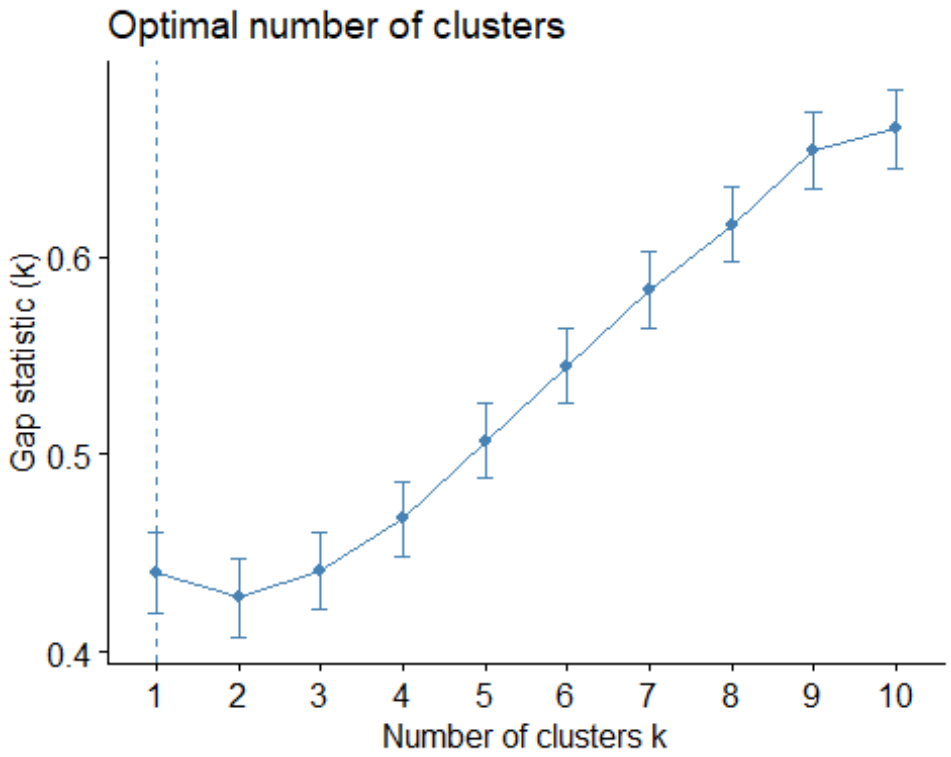
```
fviz_nbclust(df_scaled_num,kmeans,method="silhouette",nstart=50)
```




```
fviz_nbclust(df_scaled_num,kmeans,method="wss",nstart=50)
```




```
fviz_nbclust(df_scaled_num,kmeans,method="gap_stat",nboot=500,nstart=50)
```

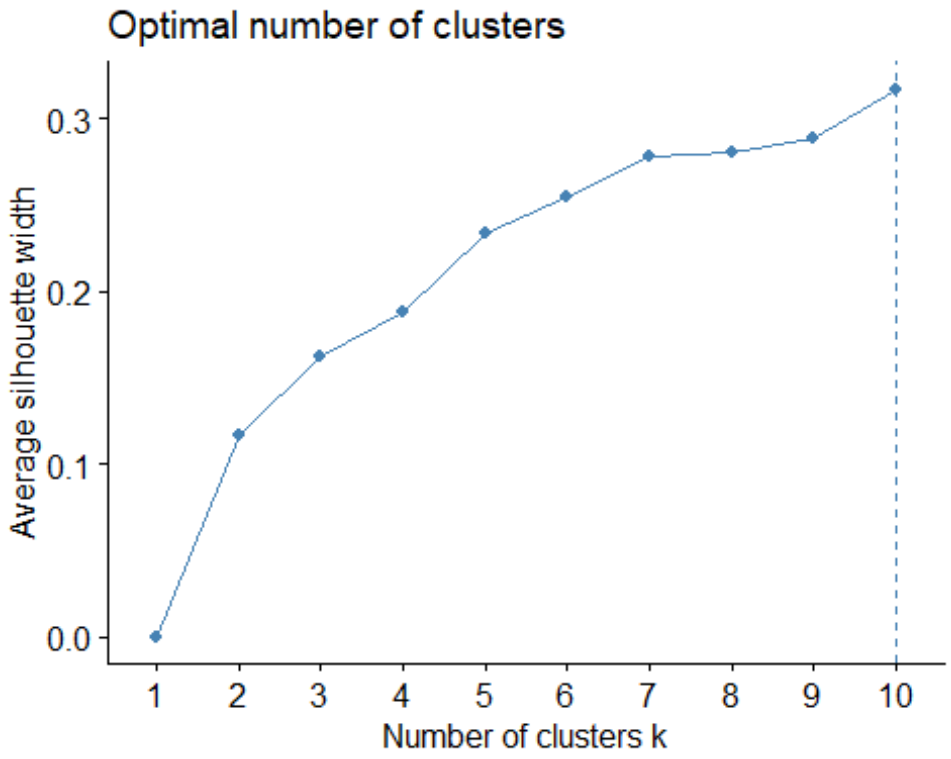


We have different results, K=10, K= 7 (there is an elbow) and K=1, the result is not clear, probably because K-means is sensible to outliers, and we found that the presence of outliers is not significantly so we now try to use the k-medoids and checking if we have better results.

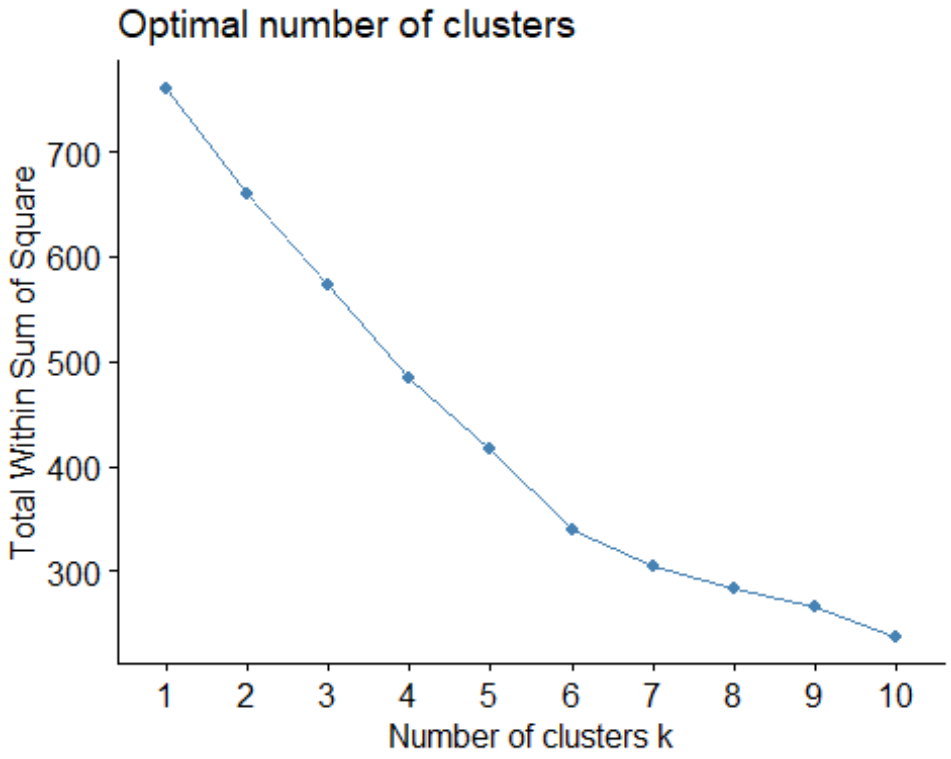
K-medoids clustering

K-medoids clustering is not limited to be used with numerical variables, each cluster is represented by one of the data points in the cluster (medoids). The most common k-medoids clustering is the PAM (partitional around medoids) algorithm. Differently from the k-means, the k-medoids works with the dissimilarity matrix D, in fact after having defined K (number of cluster) and selected randomly the k points to become the medoids, the algorithm assign each observation to the closest medoids and for each cluster search if any of the points of the cluster decreases the total dissimilarity coefficient and eventually changes the medoid.

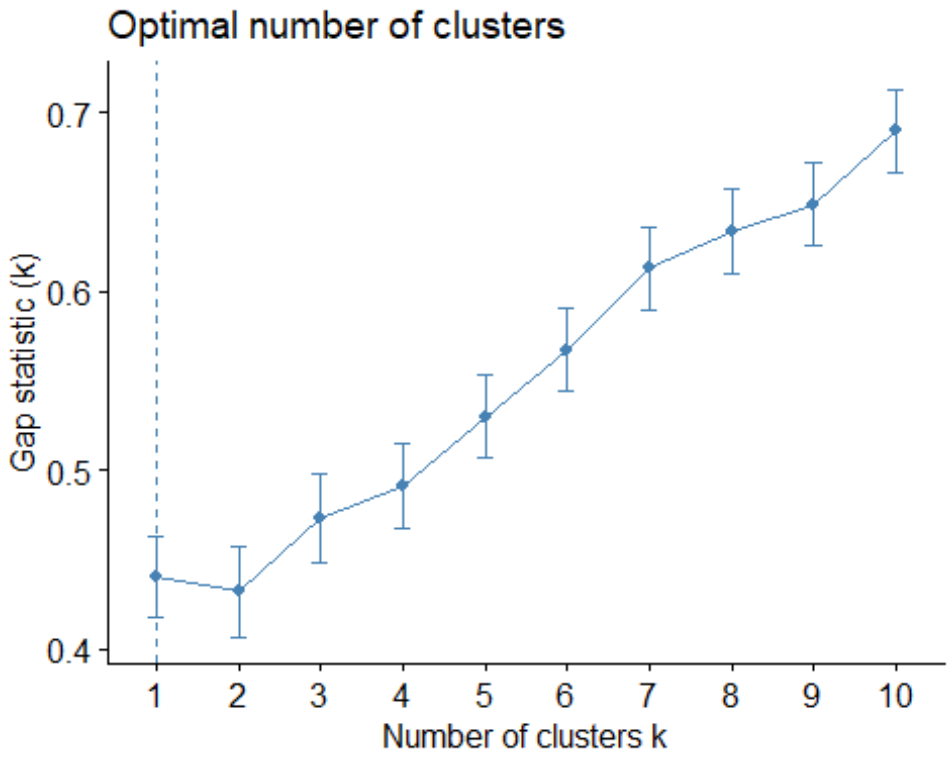
```
fviz_nbclust(df_scaled_num,cluster::pam,method =
"silhouette",metric="euclidean")
```




```
fviz_nbclust(df_scaled_num,cluster::pam,method = "wss",metric="manhattan")
```




```
fviz_nbclust(df_scaled_num,cluster::pam,method = "gap_stat",nboot = 500)
```



We can see from the plot above that the optimal K is 10, here how we can image the results seems to be more robust and give us some more information about the optimal value K.

Cluster Validation

The term cluster validation is used to design the procedure of evaluating the goodness of clustering algorithm results. It can be categorized into 3 classes: internal, external and relative cluster validation.

Internal cluster validation:

Internal validation measures often reflect the compactness, connectedness and separation of the cluster partitions. Compactness measures how close are units within the same cluster (low value indicates good compactness). Separation measures how well separated a cluster is from other cluster (high value indicates clear separation). Connectivity indicates the degree of connectedness of the cluster. It checks if neighbors are assigned to the same cluster. Generally, most of the indices for internal clustering validation combine the ratio between compactness and separation, lower value indicates good results. The two commonly used indices for assessing the goodness of clustering are the silhouette width and the Dunn index. The Dunn index is calculated as the ration between min. separation and max. diameter so it should be maximized. Cluster stability measures the robustness of clustering result by comparing it with the cluster obtained after each column (variable) is removed one at a time. APN (average proportion of non-overlap) measures the average proportion of observations not placed in the same cluster by clustering based on the data with single column removed. The AD measures the average distance between observations

placed in the same cluster under both case (full data and data without one column). The ADM measures the average distance between cluster centers for observations placed in the same cluster under both cases. The FOM measures the average intra-cluster variance of the deleted column, where the clustering is based on the remaining columns. All measures should be minimized.

##Single linkage method – Euclidean distance

```
library(clValid)

## Warning: package 'clValid' was built under R version 4.1.3

## Loading required package: cluster

clMethods<-c("hierarchical","kmeans","pam")
intern<-clValid(df_scaled_num,nClust=2:7,
                clMethods=clMethods,validation=c("internal","stability"),
                metric="euclidean",method="single")

## Warning in clValid(df_scaled_num, nClust = 2:7, clMethods = clMethods,
## validation = c("internal", : rownames for data not specified, using
1:nrow(data)

summary(intern)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6 7
##
## Validation Measures:
##                                  2       3       4       5       6       7
##
## hierarchical APN            0.0144  0.1001  0.0202  0.0134  0.0245  0.0330
##              AD             4.0117  3.8913  3.5307  3.3314  3.2745  3.1920
##              ADM            0.0638  0.5732  0.2347  0.2201  0.2589  0.2537
##              FOM            0.9937  0.9891  0.9733  0.9301  0.9316  0.8983
##              Connectivity   2.9290  7.2159 13.3802 17.3381 18.9381 21.5143
##              Dunn           0.3825  0.3113  0.3317  0.4303  0.3974  0.3929
##              Silhouette     0.3152  0.3468  0.2667  0.2881  0.2433  0.2169
## kmeans       APN            0.1623  0.1959  0.1854  0.1386  0.1405  0.1199
##              AD             3.9382  3.7270  3.4003  3.0328  2.7889  2.6005
##              ADM            0.6543  1.0948  0.9442  0.7610  0.6345  0.7671
##              FOM            0.9857  0.9610  0.9338  0.8660  0.8188  0.8071
##              Connectivity   7.2159 24.2361 24.3956 36.1540 38.0853 38.1567
##              Dunn           0.3113  0.1738  0.1507  0.1433  0.1579  0.1593
##              Silhouette     0.4030  0.2676  0.2591  0.2645  0.2915  0.3144
## pam          APN            0.1954  0.1972  0.1039  0.1336  0.1363  0.1232
##              AD             3.8617  3.4735  3.1253  2.9465  2.7142  2.4807
```

```
##              ADM            0.8388  0.8628  0.4886  0.6326  0.6096  0.5260
##              FOM            0.9762  0.9401  0.8928  0.8582  0.8373  0.7570
##              Connectivity  23.1952 27.5659 37.8159 36.9647 37.7778 46.0667
##              Dunn           0.1035  0.0853  0.0853  0.1016  0.1016  0.1037
##              Silhouette     0.1174  0.1629  0.1883  0.2340  0.2544  0.2784
##
## Optimal Scores:
##
##              Score  Method       Clusters
## APN          0.0134 hierarchical 5
## AD           2.4807 pam          7
## ADM          0.0638 hierarchical 2
## FOM          0.7570 pam          7
## Connectivity 2.9290 hierarchical 2
## Dunn         0.4303 hierarchical 5
## Silhouette   0.4030 kmeans       2
```

##Single method – Manhattan distance

```
library(clValid)
clMethods<-c("hierarchical","kmeans","pam")
intern<-clValid(df_scaled_num,nClust=2:7,
                clMethods=clMethods,validation=c("internal","stability"),
                metric="manhattan",method="single")

## Warning in clValid(df_scaled_num, nClust = 2:7, clMethods = clMethods,
## validation = c("internal", : rownames for data not specified, using
1:nrow(data)

summary(intern)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6 7
##
## Validation Measures:
##                                  2       3       4       5       6       7
##
## hierarchical APN            0.0299  0.0068  0.0320  0.0331  0.0322  0.0379
##              AD             9.6719  8.9090  8.7977  8.4117  8.0400  7.8753
##              ADM            0.2009  0.0794  0.2584  0.3420  0.2495  0.2428
##              FOM            0.9938  0.9580  0.9502  0.9127  0.9035  0.9035
##              Connectivity   2.9290  7.2159 10.1448 14.2694 18.1901 20.6901
##              Dunn           0.3068  0.2894  0.2334  0.3019  0.3438  0.3426
##              Silhouette     0.3199  0.3485  0.2680  0.2888  0.2420  0.2276
## kmeans       APN            0.1142  0.1721  0.1640  0.1431  0.2074  0.2983
##              AD             9.3104  8.4313  7.4330  6.8274  6.7456  6.5442
##              ADM            0.4674  0.7878  0.6776  0.6601  1.0364  1.2081
```

```
##              FOM            0.9812  0.9368  0.8637  0.8024  0.8213  0.8069
##              Connectivity   7.2159 23.5869 32.8448 37.6012 42.2802 44.7802
##              Dunn           0.2894  0.1376  0.1376  0.0903  0.1232  0.1232
##              Silhouette     0.4156  0.3055  0.3075  0.2961  0.2987  0.2873
## pam          APN            0.1406  0.2921  0.1650  0.2135  0.1573  0.1395
##              AD             8.8754  8.6189  7.2517  6.9219  5.9637  5.6533
##              ADM            0.4753  1.2502  0.6343  0.8934  0.5566  0.5818
##              FOM            0.9741  0.9375  0.8834  0.8758  0.7521  0.7260
##              Connectivity  36.0885 39.4488 37.1976 43.1881 47.1210 62.1214
##              Dunn           0.0497  0.0776  0.0896  0.0435  0.0525  0.1114
##              Silhouette     0.1769  0.1790  0.2439  0.2601  0.2994  0.2699
##
## Optimal Scores:
##
##              Score  Method       Clusters
## APN          0.0068 hierarchical 3
## AD           5.6533 pam          7
## ADM          0.0794 hierarchical 3
## FOM          0.7260 pam          7
## Connectivity 2.9290 hierarchical 2
## Dunn         0.3438 hierarchical 6
## Silhouette   0.4156 kmeans       2
```

##Complete linkage method – Euclidean distance

```
library(clValid)
clMethods<-c("hierarchical","kmeans","pam")
intern<-clValid(df_scaled_num,nClust=2:7,
                clMethods=clMethods,validation=c("internal","stability"),
                metric="euclidean",method="complete")

## Warning in clValid(df_scaled_num, nClust = 2:7, clMethods = clMethods,
## validation = c("internal", : rownames for data not specified, using
1:nrow(data)

summary(intern)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6 7
##
## Validation Measures:
##                                  2       3       4       5       6       7
##
## hierarchical APN            0.1929  0.1888  0.1641  0.1628  0.1582  0.2742
##              AD             4.0295  3.7446  3.5067  3.2685  3.0258  2.9406
##              ADM            0.9783  0.9877  0.9552  0.7960  0.7260  0.9276
##              FOM            0.9737  0.9317  0.8796  0.8483  0.8122  0.8057
```

```
##              Connectivity  13.8861 17.5024 28.2524 32.2103 39.6008 40.3091
##              Dunn           0.1495  0.1652  0.1702  0.2108  0.2210  0.2337
##              Silhouette     0.2300  0.2690  0.2484  0.2698  0.2701  0.2521
## kmeans       APN            0.2729  0.2436  0.2533  0.2151  0.2580  0.1518
##              AD             3.9668  3.6803  3.3932  3.0819  2.8975  2.6477
##              ADM            1.2743  1.1309  1.0583  0.9038  1.0321  0.9191
##              FOM            0.9778  0.9435  0.8564  0.8292  0.8087  0.7954
##              Connectivity  20.6552 24.2361 25.2266 35.0262 38.0853 38.1567
##              Dunn           0.1209  0.1738  0.1473  0.1579  0.1579  0.1593
##              Silhouette     0.2357  0.2676  0.2443  0.2660  0.2915  0.3144
## pam          APN            0.1954  0.1972  0.1039  0.1336  0.1363  0.1232
##              AD             3.8617  3.4735  3.1253  2.9465  2.7142  2.4807
##              ADM            0.8388  0.8628  0.4886  0.6326  0.6096  0.5260
##              FOM            0.9762  0.9401  0.8928  0.8582  0.8373  0.7570
##              Connectivity  23.1952 27.5659 37.8159 36.9647 37.7778 46.0667
##              Dunn           0.1035  0.0853  0.0853  0.1016  0.1016  0.1037
##              Silhouette     0.1174  0.1629  0.1883  0.2340  0.2544  0.2784
##
## Optimal Scores:
##
##              Score   Method       Clusters
## APN           0.1039 pam          4
## AD            2.4807 pam          7
## ADM           0.4886 pam          4
## FOM           0.7570 pam          7
## Connectivity 13.8861 hierarchical 2
## Dunn          0.2337 hierarchical 7
## Silhouette    0.3144 kmeans       7
```

##Complete linkage method – Manhattan distance

```
library(clValid)
clMethods<-c("hierarchical","kmeans","pam")
intern<-clValid(df_scaled_num,nClust=2:7,
                clMethods=clMethods,validation=c("internal","stability"),
                metric="manhattan",method="complete")

## Warning in clValid(df_scaled_num, nClust = 2:7, clMethods = clMethods,
## validation = c("internal", : rownames for data not specified, using
1:nrow(data)

summary(intern)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6 7
##
## Validation Measures:
```

```
##                                2       3       4       5       6       7
##
## hierarchical APN            0.1548  0.2570  0.2174  0.2225  0.2511  0.2099
##              AD             9.3284  8.9351  7.9782  7.3392  6.8410  6.2226
##              ADM            0.8323  1.1203  1.0388  1.2508  1.1548  1.0690
##              FOM            0.9537  0.9510  0.9119  0.8551  0.8294  0.7979
##              Connectivity  16.6167 16.7595 23.3218 40.3504 43.8960 45.0460
##              Dunn           0.1555  0.1720  0.1804  0.1226  0.1458  0.1650
##              Silhouette     0.2979  0.2909  0.3085  0.2608  0.2726  0.3065
## kmeans       APN            0.1681  0.2609  0.1990  0.2218  0.2315  0.1918
##              AD             9.1539  8.7348  7.7039  6.9675  6.4603  6.0264
##              ADM            0.6862  1.0592  0.9318  1.1003  1.0940  1.0247
##              FOM            0.9664  0.9551  0.9082  0.8510  0.8223  0.8054
##              Connectivity  20.7433 20.1571 32.8448 37.2575 47.0052 46.7306
##              Dunn           0.0649  0.0718  0.1376  0.1058  0.1353  0.1232
##              Silhouette     0.2767  0.2779  0.3075  0.2860  0.3116  0.3386
## pam          APN            0.1406  0.2921  0.1650  0.2135  0.1573  0.1395
##              AD             8.8754  8.6189  7.2517  6.9219  5.9637  5.6533
##              ADM            0.4753  1.2502  0.6343  0.8934  0.5566  0.5818
##              FOM            0.9741  0.9375  0.8834  0.8758  0.7521  0.7260
##              Connectivity  36.0885 39.4488 37.1976 43.1881 47.1210 62.1214
##              Dunn           0.0497  0.0776  0.0896  0.0435  0.0525  0.1114
##              Silhouette     0.1769  0.1790  0.2439  0.2601  0.2994  0.2699
##
## Optimal Scores:
##
##              Score   Method       Clusters
## APN           0.1395 pam          7
## AD            5.6533 pam          7
## ADM           0.4753 pam          2
## FOM           0.7260 pam          7
## Connectivity 16.6167 hierarchical 2
## Dunn          0.1804 hierarchical 4
## Silhouette    0.3386 kmeans       7
```

##Average linkage method – Euclidean

```
library(clValid)
clMethods<-c("hierarchical","kmeans","pam")
intern<-clValid(df_scaled_num,nClust=2:7,
                clMethods=clMethods,validation=c("internal","stability"),
                metric="euclidean",method="average")

## Warning in clValid(df_scaled_num, nClust = 2:7, clMethods = clMethods,
## validation = c("internal", : rownames for data not specified, using
1:nrow(data)

summary(intern)

##
## Clustering Methods:
```

```
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6 7
##
## Validation Measures:
##                                  2       3       4       5       6       7
##
## hierarchical APN            0.0127  0.0127  0.0337  0.0166  0.0586  0.1109
##              AD             3.8729  3.6680  3.5139  3.2765  3.2298  3.1068
##              ADM            0.1447  0.1363  0.2066  0.1850  0.2973  0.4580
##              FOM            0.9831  0.9525  0.8915  0.8948  0.8954  0.8868
##              Connectivity   4.2869  8.2448 13.9778 20.1421 20.2671 24.1250
##              Dunn           0.3113  0.3741  0.3444  0.3873  0.3873  0.3636
##              Silhouette     0.4181  0.3775  0.3375  0.2922  0.2654  0.2447
## kmeans       APN            0.0957  0.1789  0.1460  0.1512  0.2598  0.1031
##              AD             3.8836  3.6585  3.2889  3.0129  2.9255  2.5531
##              ADM            0.3988  0.7751  0.7343  0.7140  1.0891  0.6725
##              FOM            0.9787  0.9286  0.8746  0.8742  0.8384  0.8171
##              Connectivity   7.2159 18.0718 30.8714 36.1540 40.1444 38.1567
##              Dunn           0.3113  0.1288  0.1738  0.1433  0.1461  0.1593
##              Silhouette     0.4030  0.2430  0.2673  0.2645  0.2748  0.3144
## pam          APN            0.1954  0.1972  0.1039  0.1336  0.1363  0.1232
##              AD             3.8617  3.4735  3.1253  2.9465  2.7142  2.4807
##              ADM            0.8388  0.8628  0.4886  0.6326  0.6096  0.5260
##              FOM            0.9762  0.9401  0.8928  0.8582  0.8373  0.7570
##              Connectivity  23.1952 27.5659 37.8159 36.9647 37.7778 46.0667
##              Dunn           0.1035  0.0853  0.0853  0.1016  0.1016  0.1037
##              Silhouette     0.1174  0.1629  0.1883  0.2340  0.2544  0.2784
##
## Optimal Scores:
##
##              Score  Method       Clusters
## APN          0.0127 hierarchical 3
## AD           2.4807 pam          7
## ADM          0.1363 hierarchical 3
## FOM          0.7570 pam          7
## Connectivity 4.2869 hierarchical 2
## Dunn         0.3873 hierarchical 5
## Silhouette   0.4181 hierarchical 2
```

##Average linkage method – Manhattan

```
library(clValid)
clMethods<-c("hierarchical","kmeans","pam")
intern<-clValid(df_scaled_num,nClust=2:7,
                clMethods=clMethods,validation=c("internal","stability"),
                metric="manhattan",method="average")
```

```
## Warning in clValid(df_scaled_num, nClust = 2:7, clMethods = clMethods,
## validation = c("internal", : rownames for data not specified, using
1:nrow(data)

summary(intern)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6 7
##
## Validation Measures:
##                                  2       3       4       5       6       7
##
## hierarchical APN            0.0101  0.0408  0.1000  0.0714  0.0941  0.1286
##              AD             9.2226  8.9273  8.3493  7.4319  7.0865  6.8555
##              ADM            0.1189  0.2868  0.5674  0.6040  0.5098  0.6219
##              FOM            0.9690  0.9201  0.8970  0.8718  0.8487  0.8458
##              Connectivity   4.2869 10.0448 17.3115 25.1738 25.2738 27.6750
##              Dunn           0.2894  0.2334  0.2112  0.2768  0.2768  0.2768
##              Silhouette     0.4359  0.3350  0.3099  0.3379  0.3182  0.2860
## kmeans       APN            0.1197  0.2278  0.1114  0.1684  0.1200  0.1833
##              AD             9.3154  8.8310  7.3656  6.9106  6.2624  6.0378
##              ADM            0.4828  1.2482  0.5197  0.7534  0.8469  1.0106
##              FOM            0.9843  0.9232  0.8735  0.8339  0.8238  0.8186
##              Connectivity   7.2159 23.5869 32.8448 37.6012 40.5484 44.0940
##              Dunn           0.2894  0.1376  0.1376  0.0903  0.1466  0.1466
##              Silhouette     0.4156  0.3055  0.3075  0.2961  0.3189  0.3300
## pam          APN            0.1406  0.2921  0.1650  0.2135  0.1573  0.1395
##              AD             8.8754  8.6189  7.2517  6.9219  5.9637  5.6533
##              ADM            0.4753  1.2502  0.6343  0.8934  0.5566  0.5818
##              FOM            0.9741  0.9375  0.8834  0.8758  0.7521  0.7260
##              Connectivity  36.0885 39.4488 37.1976 43.1881 47.1210 62.1214
##              Dunn           0.0497  0.0776  0.0896  0.0435  0.0525  0.1114
##              Silhouette     0.1769  0.1790  0.2439  0.2601  0.2994  0.2699
##
## Optimal Scores:
##
##              Score  Method       Clusters
## APN          0.0101 hierarchical 2
## AD           5.6533 pam          7
## ADM          0.1189 hierarchical 2
## FOM          0.7260 pam          7
## Connectivity 4.2869 hierarchical 2
## Dunn         0.2894 hierarchical 2
## Silhouette   0.4359 hierarchical 2
```

##Ward method

```r
library(clValid)
clMethods<-c("hierarchical","kmeans","pam")
intern<-clValid(df_scaled_num,nClust=2:7,
                clMethods=clMethods,validation=c("internal","stability"),
                metric="euclidean",method="ward")
```

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

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

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

```
## Warning in clValid(df_scaled_num, nClust = 2:7, clMethods = clMethods,
## validation = c("internal", : rownames for data not specified, using
1:nrow(data)
```

```
summary(intern)
```

```
##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6 7
##
## Validation Measures:
##                                 2       3       4       5       6       7
##
## hierarchical APN           0.1509  0.3058  0.2962  0.2976  0.2192  0.1891
##              AD            3.7933  3.6296  3.3766  3.1635  2.8281  2.5971
##              ADM           0.5769  1.2728  1.3355  1.3437  1.0310  0.8586
##              FOM           0.9890  0.9459  0.9008  0.8943  0.8233  0.7652
##              Connectivity  11.4119 20.9845 26.4175 29.5754 36.4329 44.8429
```

```
##              Dunn           0.0892  0.1035  0.1092  0.1144  0.1415  0.1593
##              Silhouette     0.0958  0.1536  0.2020  0.2413  0.2753  0.2826
## kmeans       APN            0.2973  0.2554  0.2485  0.2648  0.2165  0.1962
##              AD             3.9336  3.5476  3.2638  3.0721  2.7929  2.5955
##              ADM            1.2556  1.1976  1.1349  1.1856  0.9837  0.8840
##              FOM            0.9943  0.9645  0.9089  0.9011  0.8173  0.7708
##              Connectivity  18.5313 24.3790 29.8706 30.1567 40.2028 44.2734
##              Dunn           0.1030  0.1035  0.1274  0.1431  0.1650  0.1593
##              Silhouette     0.1354  0.1760  0.2211  0.2589  0.2840  0.2896
## pam          APN            0.1954  0.1972  0.1039  0.1336  0.1363  0.1232
##              AD             3.8617  3.4735  3.1253  2.9465  2.7142  2.4807
##              ADM            0.8388  0.8628  0.4886  0.6326  0.6096  0.5260
##              FOM            0.9762  0.9401  0.8928  0.8582  0.8373  0.7570
##              Connectivity  23.1952 27.5659 37.8159 36.9647 37.7778 46.0667
##              Dunn           0.1035  0.0853  0.0853  0.1016  0.1016  0.1037
##              Silhouette     0.1174  0.1629  0.1883  0.2340  0.2544  0.2784
##
## Optimal Scores:
##
##              Score   Method       Clusters
## APN           0.1039 pam          4
## AD            2.4807 pam          7
## ADM           0.4886 pam          4
## FOM           0.7570 pam          7
## Connectivity 11.4119 hierarchical 2
## Dunn          0.1650 kmeans       6
## Silhouette    0.2896 kmeans       7
```

In general the value of K=2 is the most chosen using the k-means method and hierarchical method, while K=7 is the other value that we find several times using the pam method. This result confirms the previous analysis in fact those 2 values are the most frequently suggested by the algorithms.

So now we compute a hierarchical clustering, and we cut it with K=2 clusters.

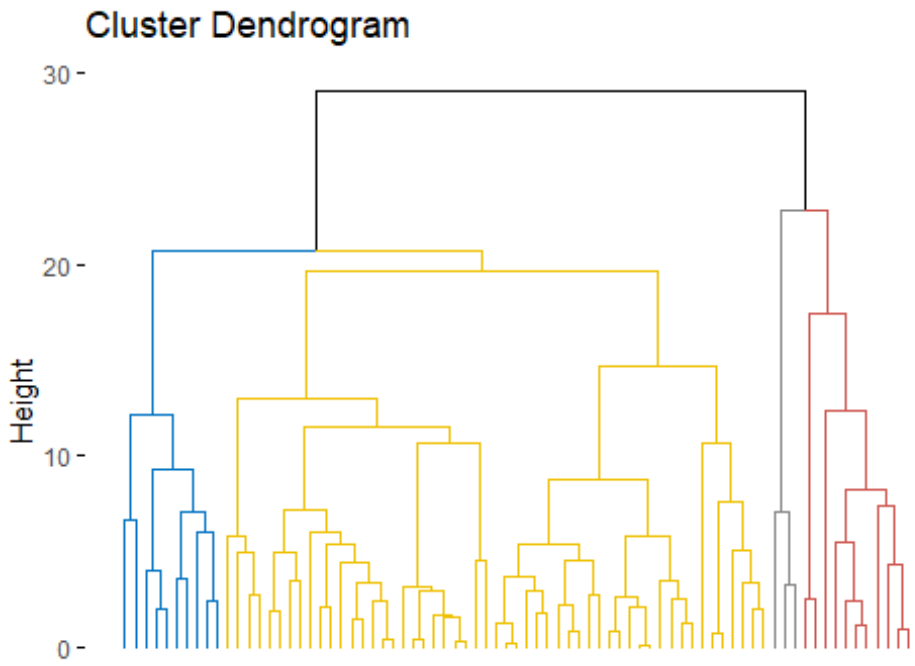
I will show using the Silhouette Width the difference using K=4 and K=2 that is the suggested value:

```
library(eclust)

## Warning: package 'eclust' was built under R version 4.1.3

## See example usage at http://sahirbhatnagar.com/eclust/

install.packages('eclust')

## Warning: package 'eclust' is in use and will not be installed

library(eclust)
hc.res <- eclust(df_scaled_num, k=4, "hclust", hc_metric= "manhattan",
hc_method = "complete", nboot=10)
```

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

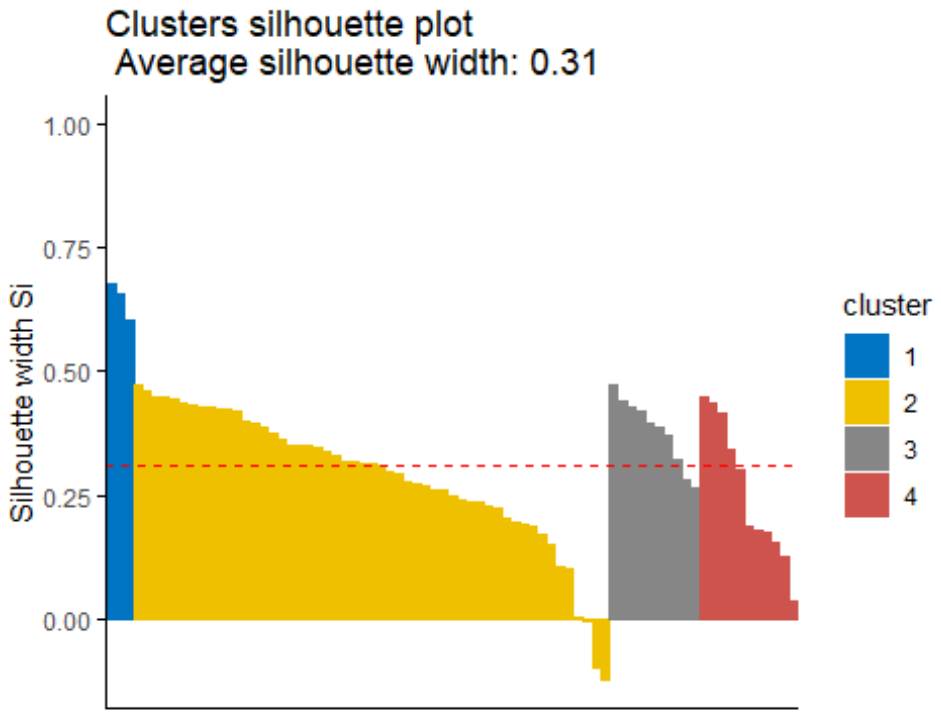
```
fviz_dend(hc.res, show_labels = FALSE, palette="jco", as.ggplot=TRUE)
```

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




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

```
##   cluster size ave.sil.width
## 1       1    3          0.65
## 2       2   53          0.29
## 3       3   10          0.38
## 4       4   11          0.25
```



We have an average silhouette width equals to 0.27. Cluster 2 is that one with the greatest number of units. We can see that cluster 1,3 and cluster 4 have a silhouette width higher than the average,while in cluster 2 just some units are higher than the average. In cluster 2 some units are a negative silhouette width indicates that units are clustered not in the right way.

```
sil<-hc.res$silinfo$width
neg_sil_index<-which(sil[,"sil_width"]<0)
sil[neg_sil_index, ,drop=FALSE]
```

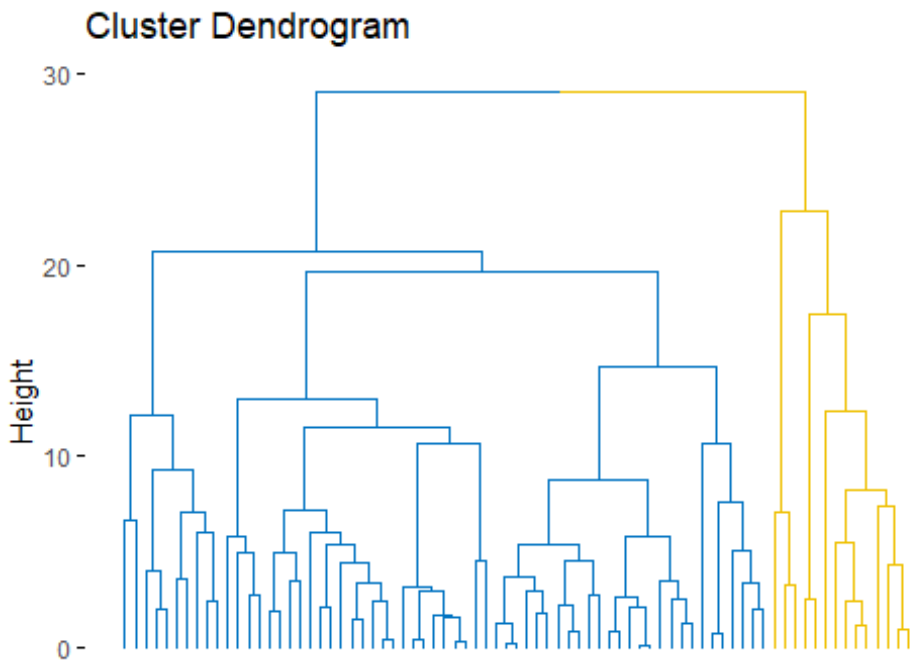
```
##    cluster neighbor    sil_width
## 51       2        4 -0.006185248
## 45       2        3 -0.100801556
## 46       2        3 -0.123924553
```

As we can see the 1 unit clustered in cluster 2 according to the silhouette width belongs to cluster 4 and the 2 units clustered in cluster 2 belongs to cluster 3.

Now i want to apply the same method and cut the dendogram with the optimal value suggested k=2:

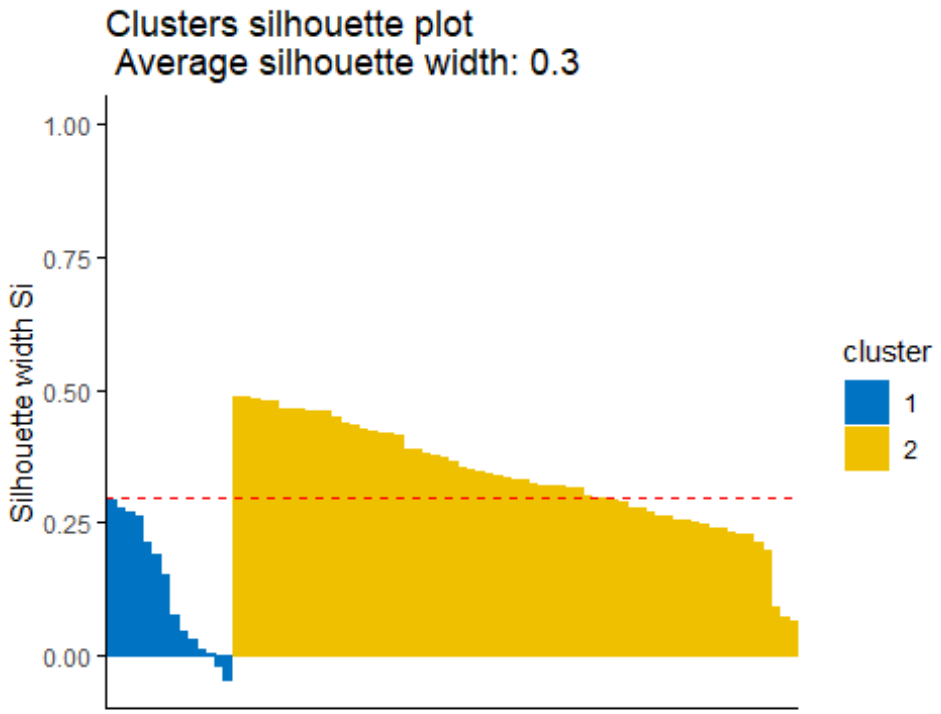
```
hc.res<-eclust(df_scaled_num,"hclust",k=2,hc_metric = "manhattan",hc_method =
"complete",graph=FALSE)
fviz_dend(hc.res,show_labels = FALSE, palette="jco",as.ggplot=TRUE)
```

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




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

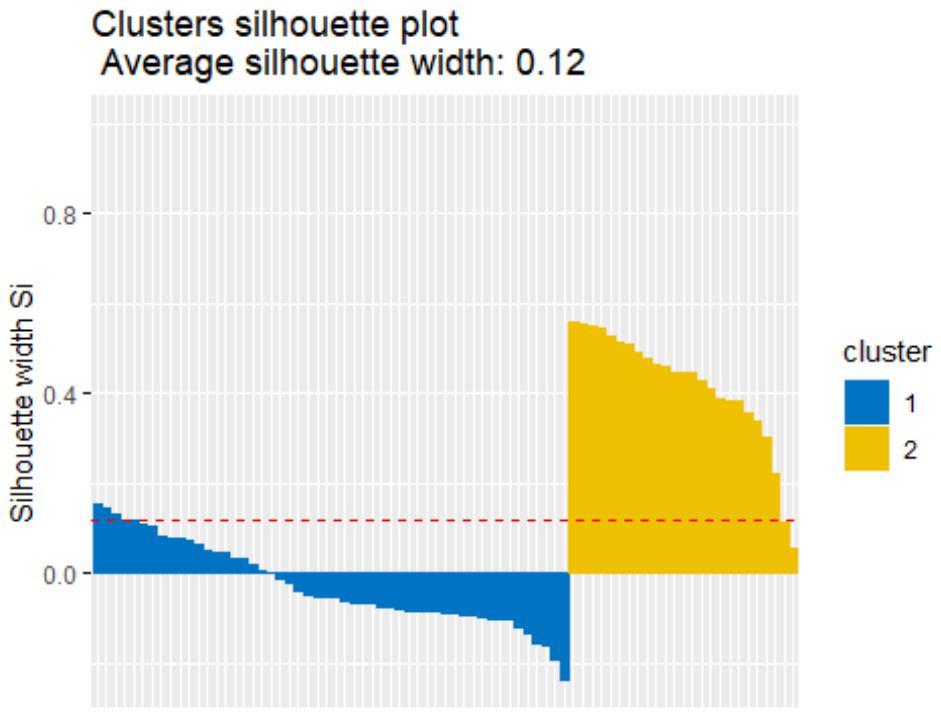
```
##   cluster size ave.sil.width
## 1       1   14          0.13
## 2       2   63          0.34
```



As we can see there are some units clustered in 1 cluster in a wrong way, and 63 units out 80 are in cluster 2. The just 80 units available in the data set are not enough to explain our clustering in the right way. We can also try to use the pam with K=2

```
km.res<-eclust(df_scaled_num,"pam",k=2,graph=FALSE,hc_metric="euclidean")
fviz_silhouette(km.res,palette="jco",as.ggtheme=theme_classic())
```

```
##   cluster size ave.sil.width
## 1       1   52          -0.03
## 2       2   25           0.42
```



We can see that the pam algorithm with 2 clusters spread 52 units in the first cluster and 25 units in the other cluster, but as we can see the average silhouette width is equal to 0.12, much less than 0.46 than the other representation, most of the data in cluster 1 are under the average and there are some values under the 0 value, this means that are wrong clustered.

##Dunn Index

The Dunn index is an internal clustering validation measure which can be computed in a)"Separation part" and b)"Compactness part", for each cluster is evaluated the distance between each of the unit in the cluster and the units in the other clusters and use the minimum distance (a), and compute the distance between the units in the same cluster considering the maximum (b), in our case Dunn is equal to 0.226, that is a good value, so this means that units are good clustered:

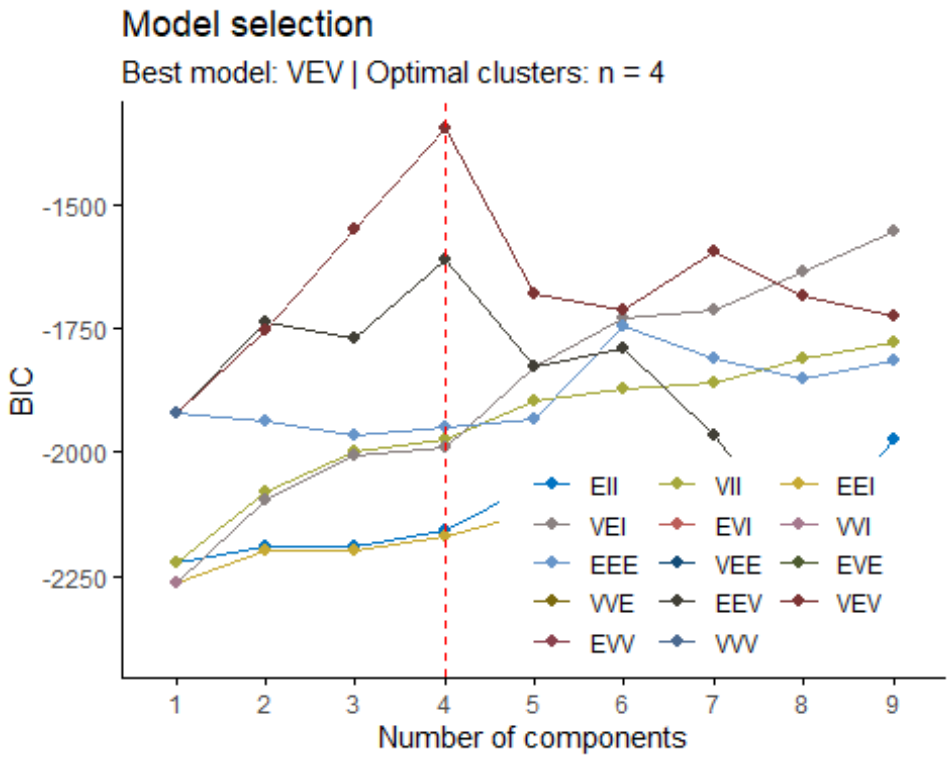
```
library(fpc)

## Warning: package 'fpc' was built under R version 4.1.3

hc_stats<-cluster.stats(dist(df_scaled_num),hc.res$cluster)
hc_stats$dunn

## [1] 0.2269196
```

##Model – Based Clustering

The model- based clustering considers the sample observations as coming from a distribution that is mixture of two or more, say K, distributions, commonly one for each cluster. Each component is described by a probability density or mass function and has an

associated probability in the mixture. This model uses a soft assignment where each data point has a probability of belonging to each cluster. If the unit has a higher probability to belong to a defined cluster, it will be assigned to it according to the maximum a posteriori probability criterion. An important class of mixture of distributions concerns mixture of ddimensional Gaussian distributions, which are usually referred to a Gaussian mixtures or Normal mixtures. Gaussian mixtures are the most used because they can model a great variety of density functions. By selecting suitable parameters. Now we are implementing the Model – based clustering via Parsimonious Gaussian mixtures model, according to the parsimony criterion (using a low number of parameters).

```
library(mclust)

## Warning: package 'mclust' was built under R version 4.1.3

## Package 'mclust' version 5.4.10
## Type 'citation("mclust")' for citing this R package in publications.

##
## Attaching package: 'mclust'

## The following object is masked from 'package:psych':
##
##     sim

## The following object is masked from 'package:mgcv':
##
##     mvn

## The following object is masked from 'package:gamlss.data':
##
##     acidity

library(ggrepel)

## Warning: package 'ggrepel' was built under R version 4.1.3

mod<-Mclust(df_scaled_num)

fviz_mclust(mod,"BIC",palette="jco")
```




```
summary(mod)

## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEV (ellipsoidal, equal shape) model with 4 components:
##
##  log-likelihood  n  df       BIC       ICL
##       -161.8172 77 236 -1348.772 -1348.777
##
## Clustering table:
##  1  2  3  4
## 23 19 25 10
```

The best model configuration is VEV (variable volume, equal shape, variable orientation). We can see looking at the information contained in the model that the BIC (Bayesian Information Criterion) has a log-likelihood of -161.81, it uses 236 parameters, the BIC value is -1348.772 and the ICL (Integrated completed Likelihood) is -1348.777. We have 4 clusters.

The three best BIC values:

```
summary(mod$BIC)

## Best BIC values:
##              VEV,4      VEV,3      VEI,9
```

```
## BIC      -1348.772 -1551.1680 -1554.8499
## BIC diff     0.000  -202.3955  -206.0774
```

This means that the best 3 models according to the BIC is VEI (variable volume, equal shape, variable orientation) with 9 clusters, the second one is VEV with 3 clusters and the third one is VEV with 4 clusters. the plot below is the graphical representation of BIC, we have different curves for each number of cluster and for each parsimonious configuration we have a different symbol.

```
mod$G
```

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

```
head(round(mod$z,6),20)
```

```
##           [,1]     [,2]     [,3] [,4]
##  [1,] 0.000000 0.000000 1.000000    0
##  [2,] 1.000000 0.000000 0.000000    0
##  [3,] 0.999753 0.000000 0.000247    0
##  [4,] 0.999977 0.000000 0.000023    0
##  [5,] 0.000000 1.000000 0.000000    0
##  [6,] 0.000000 0.999924 0.000076    0
##  [7,] 0.000000 1.000000 0.000000    0
##  [8,] 1.000000 0.000000 0.000000    0
##  [9,] 0.000001 0.000000 0.999999    0
## [10,] 0.000000 0.000000 1.000000    0
## [11,] 0.000000 1.000000 0.000000    0
## [12,] 1.000000 0.000000 0.000000    0
## [13,] 0.000000 1.000000 0.000000    0
## [14,] 0.000000 0.000000 1.000000    0
## [15,] 0.000000 0.999995 0.000005    0
## [16,] 0.000000 0.000000 1.000000    0
## [17,] 0.000000 0.000000 1.000000    0
## [18,] 0.000000 1.000000 0.000000    0
## [19,] 0.000000 0.999971 0.000029    0
## [20,] 0.000000 0.000000 1.000000    0
```

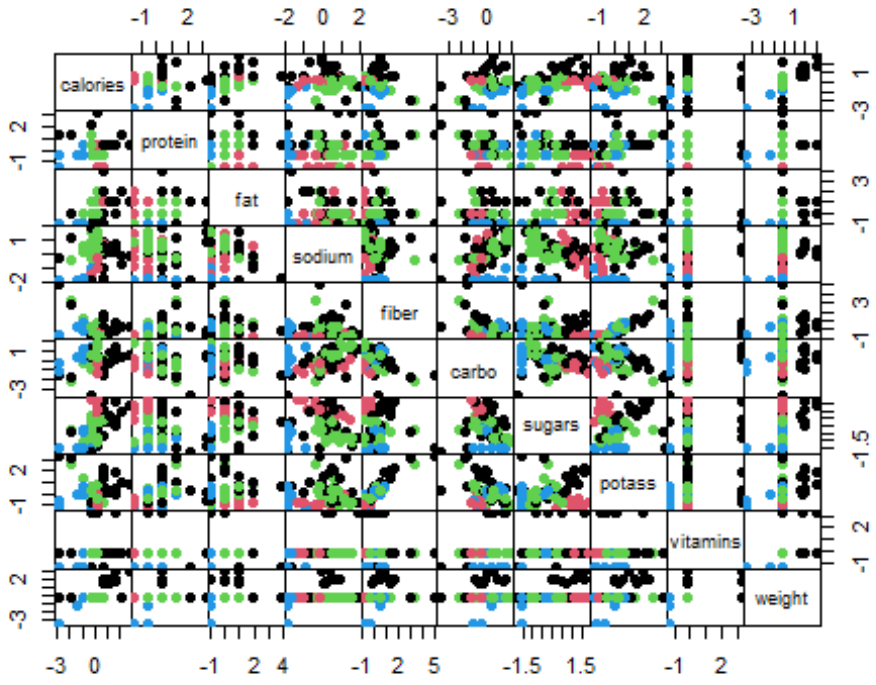
This is z, the matrix of posterior probabilities, that one of soft assignment, and we can see that each unit belonging to its cluster according to a defined probability, for example on number 12 is complete 1% of probability to belong in cluster 1 while it has a zero percentage belonging in cluster 2. We can also see a hard clustering that is obtained from a fuzzy clustering based on the maximum a posteriori probability (MAP) criterion:

```
head(mod$classification,20)
```

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

Now we can graphically visualize the clustering results:

```
pairs(df_scaled_num, gap = 0, pch=16, col=mod$classification)
```


calories
protein
fat
sodium
fiber
carbo
sugars
potass
vitamins
weight