2021

UNIVERSITÀ
degli STUDI
di CATANIA

Report: Video Games Sales Dataset

DATA ANALYSIS AND STATISTICAL LEARNING

PROF. A. PUNZO

THAMIRES DE SOUZA OLIVEIRA | 1000023616

Summary

Report: Video Games Sales Dataset

Data Description

The dataset chosen for this analysis was taken directly from the Kaggle website (https://www.kaggle.com/) and was generated from a scrap from http://www.vgchartz.com/. The database contains a list of video games with sales of over 100,000 copies put in a ranking of historical sales until the year of 2016.

Fields include

Name - The games name

Platform - Platform of the games release (i.e. PC,PS4, etc.)

Year - Year of the game's release

Genre - Genre of the game

Publisher - Publisher of the game

NA_Sales - Sales in North America (in millions)

EU_Sales - Sales in Europe (in millions)

JP_Sales - Sales in Japan (in millions)

Other_Sales - Sales in the rest of the world (in millions)

Global_Sales - Total worldwide sales.

Overall Dataset

The original dataset contains 16.598 observations, but I will import it with only the first 200 top ranking observations to optimize our analysis.

```
library(readxl)
vg_sales <- read_excel("vg_sales.xlt",
    col_types = c("numeric", "text", "text",
        "numeric", "text", "text", "numeric",
        "numeric", "numeric", "numeric",
        "numeric"), n_max = 200)

str(vg_sales)
```

```
## tibble [200 x 11] (S3: tbl_df/tbl/data.frame)
##  $ Rank         : num [1:200] 1 2 3 4 5 6 7 8 9 10 ...
```

```
##  $ Name        : chr [1:200] "Wii Sports" "Super Mario Bros." "Mario K
art Wii" "Wii Sports Resort" ...
##  $ Platform    : chr [1:200] "Wii" "NES" "Wii" "Wii" ...
##  $ Year        : num [1:200] 2006 1985 2008 2009 1996 ...
##  $ Genre       : chr [1:200] "Sports" "Platform" "Racing" "Sports" ...
##  $ Publisher   : chr [1:200] "Nintendo" "Nintendo" "Nintendo" "Nintend
o" ...
##  $ NA_Sales    : num [1:200] 41.5 29.1 15.8 15.8 11.3 ...
##  $ EU_Sales    : num [1:200] 29.02 3.58 12.88 11.01 8.89 ...
##  $ JP_Sales    : num [1:200] 3.77 6.81 3.79 3.28 10.22 ...
##  $ Other_Sales : num [1:200] 8.46 0.77 3.31 2.96 1 0.58 2.9 2.85 2.26
0.47 ...
##  $ Global_Sales: num [1:200] 82.7 40.2 35.8 33 31.4 ...
```

When importing the data, the dataset contained 7 numeric variables, 4 categorical variables and 200 observations. The first step is to transform the *Rank* column into row names:

```
for (i in 1:nrow(vg_sales)){
  rownames(vg_sales)[i]<- vg_sales[i,1]
}

vg_sales <- vg_sales[,-1]
```

Then we have to change the variables *Platform*, *Genre* and *Publisher* into factors:

```
vg_sales$Platform = factor(vg_sales$Platform)
vg_sales$Genre = factor(vg_sales$Genre)
vg_sales$Publisher = factor(vg_sales$Publisher)
```

Then we have to check if we have any NA values.

```
any(is.na(vg_sales))

## [1] FALSE
```

As we checked, the data set doesn't have any NA values and with this we have 7 numerical variables, 4 categorical variables and 3 of it structured as factors.

```
str(vg_sales)

## tibble [200 x 10] (S3: tbl_df/tbl/data.frame)
##  $ Name        : chr [1:200] "Wii Sports" "Super Mario Bros." "Mario K
art Wii" "Wii Sports Resort" ...
##  $ Platform    : Factor w/ 21 levels "2600","3DS","DS",..: 17 9 17 17
4 4 3 17 17 9 ...
##  $ Year        : num [1:200] 2006 1985 2008 2009 1996 ...
##  $ Genre       : Factor w/ 12 levels "Action","Adventure",..: 11 5 7 1
1 8 6 5 4 5 9 ...
##  $ Publisher   : Factor w/ 23 levels "505 Games","Activision",..: 12 1
```

```
2 12 12 12 12 12 12 12 ...
##  $ NA_Sales    : num [1:200] 41.5 29.1 15.8 15.8 11.3 ...
##  $ EU_Sales    : num [1:200] 29.02 3.58 12.88 11.01 8.89 ...
##  $ JP_Sales    : num [1:200] 3.77 6.81 3.79 3.28 10.22 ...
##  $ Other_Sales : num [1:200] 8.46 0.77 3.31 2.96 1 0.58 2.9 2.85 2.26
0.47 ...
##  $ Global_Sales: num [1:200] 82.7 40.2 35.8 33 31.4 ...
```

Therefore we have a final *vg_sale* dataset of 200 observations and 10 variables, where each row represents a ranking place on Video Games sales all over the world and this will be the dataset we will use on this analysis. Check the details below:

```
vg_sales
```

```
## # A tibble: 200 x 10
##    Name  Platform  Year Genre Publisher NA_Sales EU_Sales JP_Sales Oth
er_Sales
##    <chr> <fct>    <dbl> <fct> <fct>        <dbl>    <dbl>    <dbl>
<dbl>
##  1 Wii ~ Wii       2006 Spor~ Nintendo      41.5    29.0      3.77
8.46
##  2 Supe~ NES       1985 Plat~ Nintendo      29.1     3.58     6.81
0.77
##  3 Mari~ Wii       2008 Raci~ Nintendo      15.8    12.9      3.79
3.31
##  4 Wii ~ Wii       2009 Spor~ Nintendo      15.8    11.0      3.28
2.96
##  5 Poke~ GB        1996 Role~ Nintendo      11.3     8.89    10.2
1
##  6 Tetr~ GB        1989 Puzz~ Nintendo      23.2     2.26     4.22
0.580
##  7 New ~ DS        2006 Plat~ Nintendo      11.4     9.23     6.5
2.9
##  8 Wii ~ Wii       2006 Misc  Nintendo      14.0     9.2      2.93
2.85
##  9 New ~ Wii       2009 Plat~ Nintendo      14.6     7.06     4.7
2.26
## 10 Duck~ NES       1984 Shoo~ Nintendo      26.9     0.63     0.28
0.47
## # ... with 190 more rows, and 1 more variable: Global_Sales <dbl>
```

Now that we optimized and arranged the datas, we can start the analysis.

Univariate Analysis

First of all, lets analyze every single one of the variables, to describe the data and maybe find patterns with it.

Name

Name is a categorical variable, so in this case we can only visualize its lenght, class and mode.

```
summary(vg_sales$Name)

##    Length     Class      Mode
##       200 character character
```

Platform

Platform is a factor, so we can check how many levels it has and as we can see below it has 21 levels, or we can say that we have 21 different platforms in the first 200 video games in the rank. The platform X360 and PS3 has equally 28 video games represented in this dataset.

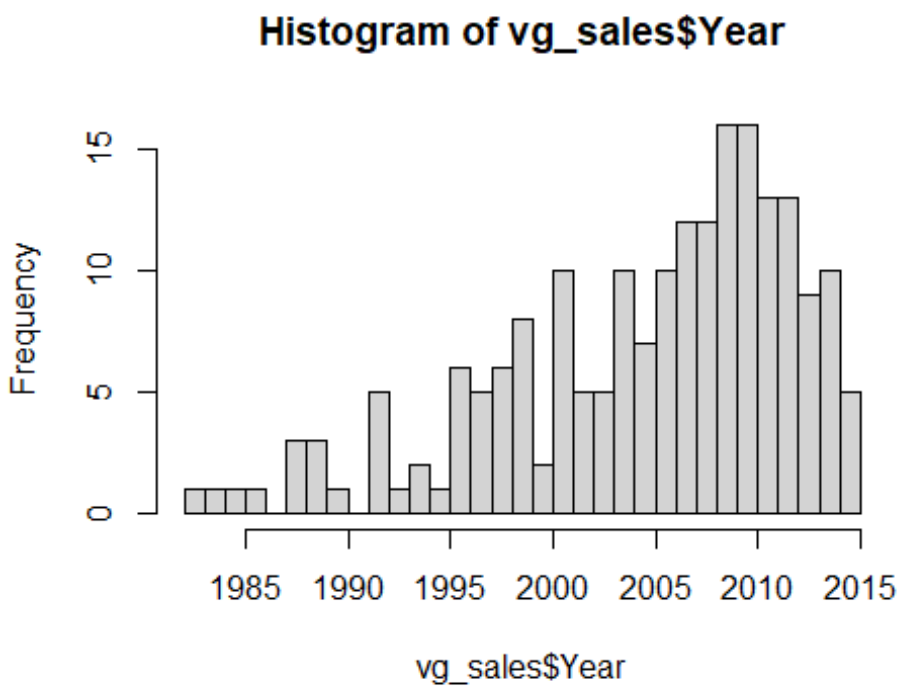
```
summary(vg_sales$Platform)

## 2600  3DS   DS   GB  GBA   GC  GEN  N64  NES   PC   PS  PS2  PS3  PS4
PSP SNES
##    1    8   19   11    7    3    1    7    6    4   14   18   28    8
3    6
##  Wii WiiU X360   XB XOne
##   21    2   28    2    3

levels(vg_sales$Platform)

##  [1] "2600" "3DS"  "DS"   "GB"   "GBA"  "GC"   "GEN"  "N64"  "NES"  "P
C"
## [11] "PS"   "PS2"  "PS3"  "PS4"  "PSP"  "SNES" "Wii"  "WiiU" "X360" "X
B"
## [21] "XOne"
```

Year

Year is a discrete random variable, so we can draw a histogram to visualize better the frequencies of years in this dataset. As we can see below, in the years of 2010 and 2011 we had more video games in the top 200 rank of sales. Another thing interesting in this is that we can see that we dont have any video games of the years 1986 and 1990 into this dataset.

```
summary(vg_sales$Year)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1982    2001    2007    2005    2010    2015
```

```
hist(vg_sales$Year, breaks = 30)
```



Genre

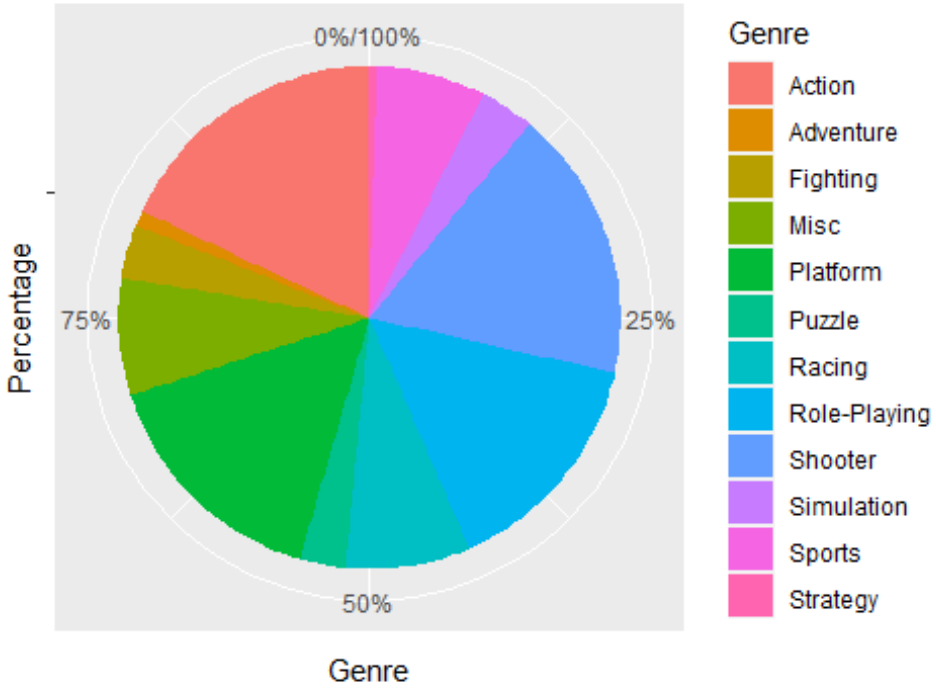
As well as *Platform*, *Genre* is also a factor, with that we can check how many levels this factor has and as we can see below, there are 12 levels in the *Genre* factor, so we can assume that this dataset has 12 types of video game genres in the Top 200 rank of sales. And the genre Action followed by Shooter are the ones that appears the most in that rank.

```
summary(vg_sales$Genre)

##       Action    Adventure     Fighting         Misc     Platform
Puzzle
##           36            2            7           15           31
6
##       Racing Role-Playing      Shooter   Simulation       Sports     S
trategy
##           16           30           35            7           14
1

levels(vg_sales$Genre)

##  [1] "Action"       "Adventure"    "Fighting"     "Misc"         "Plat
form"
##  [6] "Puzzle"       "Racing"       "Role-Playing" "Shooter"      "Simu
```

```
lation"
## [11] "Sports"       "Strategy"
```

In this case, *Genre* doesn't have as many levels as *Platform* so we can draw a pie graph to help us to visualize better the distribuition of genres in the top rank:

```
library(ggplot2)

ggplot(subset(vg_sales), aes(y="", fill=Genre)) +
geom_bar(aes(x=..count../sum(..count..))) +
scale_x_continuous(labels=scales::percent) +
ylab('Percentage') +
xlab('Genre') +
coord_polar()
```



Publisher

Publisher is also a factor, and in this case with 23 levels. The publisher with more video games represented into this dataset is *Nintendo* with 80 units, almost the half of all observations.

```
summary(vg_sales$Publisher)

##                                505 Games                                Act
ivision
##                                        2
21
##                                    Atari                        Bethesda So
```

```
ftworks
##                                     1                                 
3
##                                 Capcom                      Eidos Inte
ractive
##                                     3                                 
1
##                         Electronic Arts                         GT Inte
ractive
##                                    17                                 
1
##           Konami Digital Entertainment                           Lu
casArts
##                                     3                                 
1
##                  Microsoft Game Studios                            N
intendo
##                                    15                                 
80
##                               RedOctane                              
Sega
##                                     1                                 
3
##             Sony Computer Entertainment     Sony Computer Entertainment
Europe
##                                    17                                 
1
##                             Square Enix                           Squ
areSoft
##                                     3                                 
2
##                   Take-Two Interactive                              
Ubisoft
##                                    13                                 
9
##                   Universal Interactive                    Virgin Inte
ractive
##                                     1                                 
1
## Warner Bros. Interactive Entertainment
##                                     1

levels(vg_sales$Publisher)

##  [1] "505 Games"
##  [2] "Activision"
##  [3] "Atari"
##  [4] "Bethesda Softworks"
##  [5] "Capcom"
##  [6] "Eidos Interactive"
```

```
##  [7] "Electronic Arts"
##  [8] "GT Interactive"
##  [9] "Konami Digital Entertainment"
## [10] "LucasArts"
## [11] "Microsoft Game Studios"
## [12] "Nintendo"
## [13] "RedOctane"
## [14] "Sega"
## [15] "Sony Computer Entertainment"
## [16] "Sony Computer Entertainment Europe"
## [17] "Square Enix"
## [18] "SquareSoft"
## [19] "Take-Two Interactive"
## [20] "Ubisoft"
## [21] "Universal Interactive"
## [22] "Virgin Interactive"
## [23] "Warner Bros. Interactive Entertainment"
```

In this case I didnt draw a pie graph cause it has many levels and the visualization would be not so clarifying.

NA_Sales

NA_Sales is a continuous variable that represents the video games sales (in milions) in North America. We can check some values from this sample using the summary function.

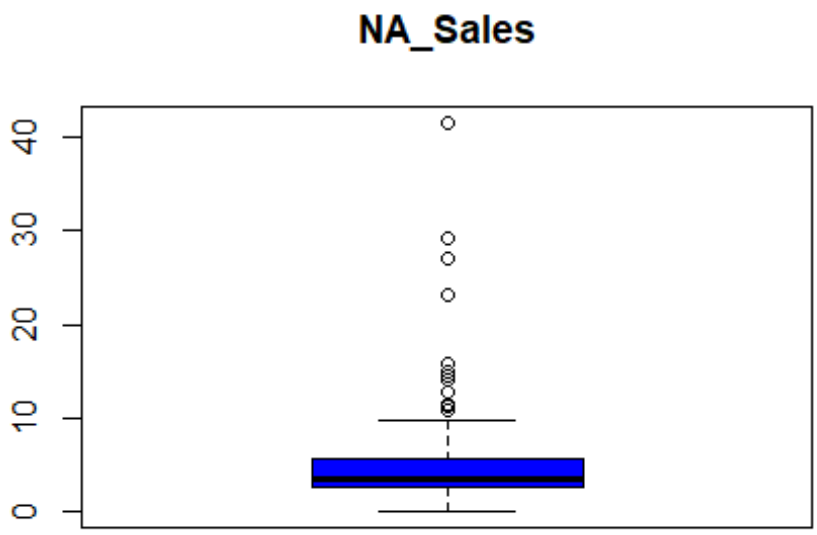
We can see that the mean of sales in North America is 4.892 millions, the maximun value is 41.49 milions and the minimun value is 0.070 milions. Other values that we have with this function are the 1st and 3rd quartil and the median, as you can see below:

```
summary(vg_sales$NA_Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.070   2.638   3.565   4.892   5.628  41.490
```

Since *NA_Sales* is a continuous variable, we will use a boxplot view, which is a graphical and compact way to analyze a variable.

```
library(ggplot2)
boxplot(vg_sales$NA_Sales, main="NA_Sales", col="blue")
```

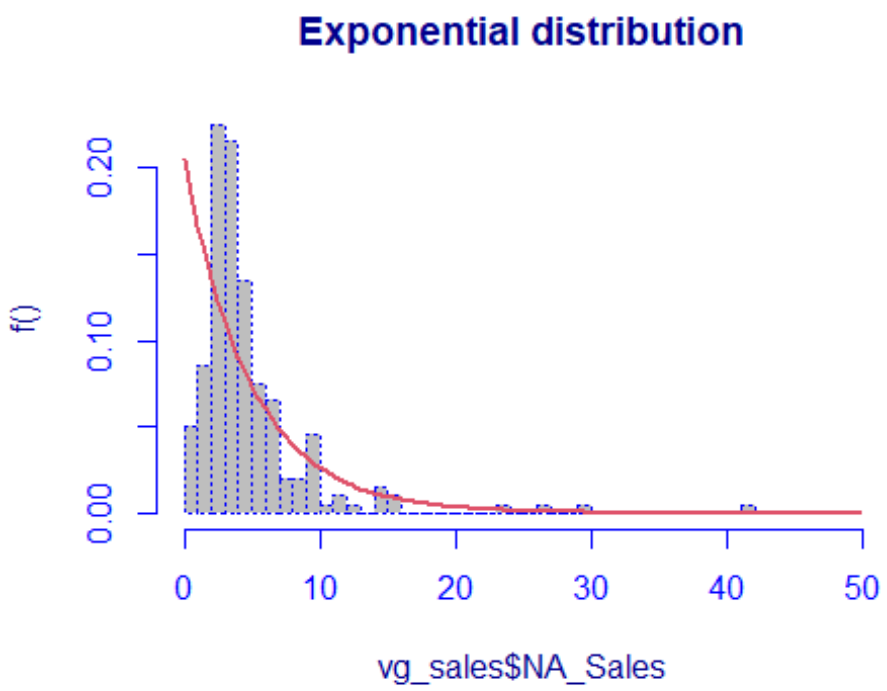


In the boxplot we can see some graphical representation of the values we have seen before in the summary like:

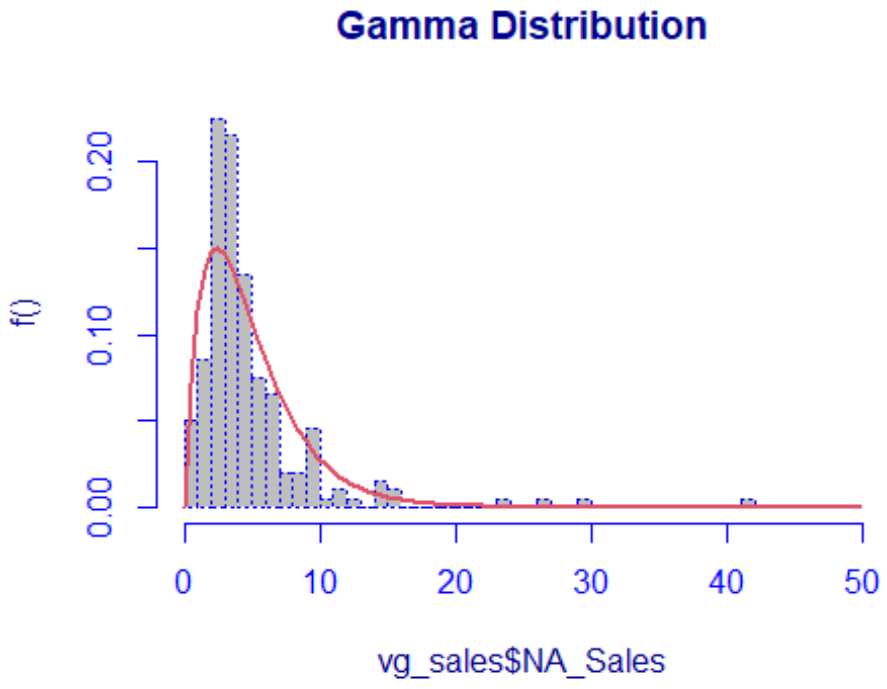
- The bottom segment of the box represents the first quartile and how we have checked before, is 2.638 milions;
- The segment in the middle of the box represents the median value (second quartile) that is 3.565 milions;
- The top segment of the box represents the third quartile that is 5.628 milions;
- The bottom whisker indicates the minimun value of the sample (with no outliers) that is we can see that is close to 0;
- The top whisker indicates the maximun value of the sample (with no outliers) and we can check that is close to 10;
- The dots represent the values of the sample which are behind the limits defined by the whiskers. These are usually referred to as outliers and considered as abnormal values in the sample and in this case goes to 10 milion until more than 40 milions sales.

Now we can look for the model that fits best for the distribution:

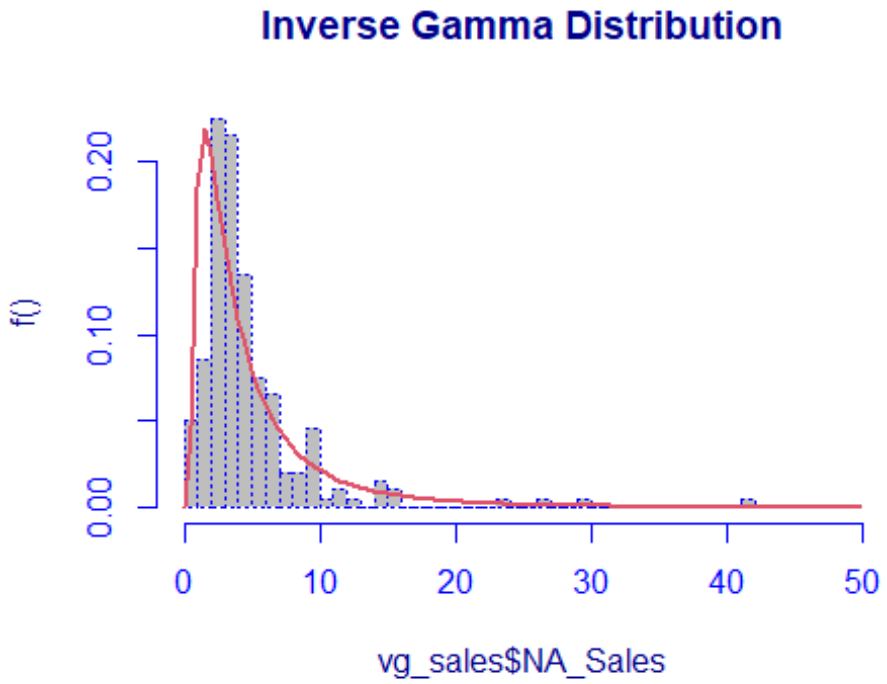
```
library(gamlss)

fit.EXP <- histDist(vg_sales$NA_Sales, family=EXP, nbins = 30, main="Expo
nential distribution")
```



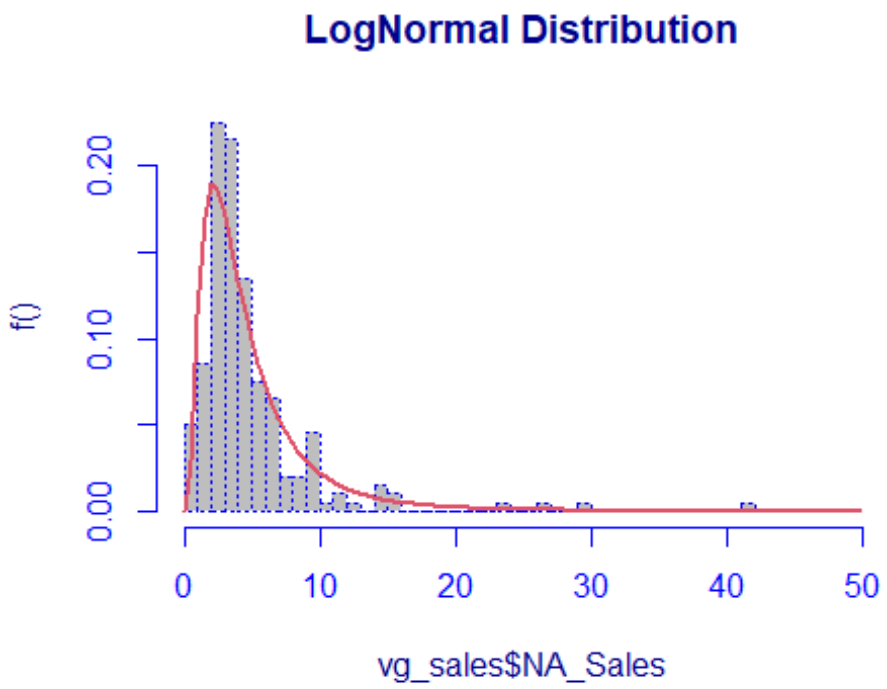

```
fit.GA <- histDist(vg_sales$NA_Sales, family=GA, nbins=30, main = "Gamma
Distribution")
```



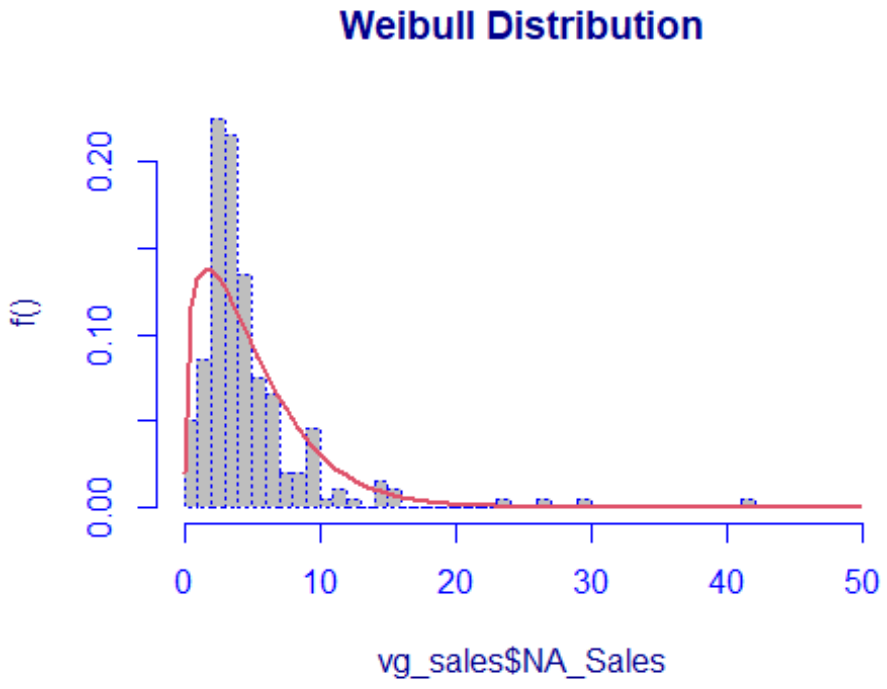
```
fit.IG <- histDist(vg_sales$NA_Sales, family=IG, nbins=30, main = "Invers
e Gamma Distribution")
```




```
fit.LOGNO <- histDist(vg_sales$NA_Sales, family=LOGNO, nbins=30, main = "
LogNormal Distribution")
```




```
fit.WEI <- histDist(vg_sales$NA_Sales,family=WEI, nbins=30, main = "Weibu
ll Distribution")
```



```
data.frame(row.names = c("Exponential", "Gamma", "Inverse Gaussian", "Log
-Normal", "Weibull"),
AIC=c(AIC(fit.EXP), AIC(fit.GA), AIC(fit.IG), AIC(fit.LOGNO), AIC(fit.WEI
)),
SBC=c(fit.EXP$sbc, fit.GA$sbc, fit.IG$sbc, fit.LOGNO$sbc, fit.WEI$sbc))
```

```
##                         AIC       SBC
## Exponential      1037.0078 1040.3061
## Gamma             994.8382 1001.4348
## Inverse Gaussian 1038.9999 1045.5966
## Log-Normal        979.6925  986.2891
## Weibull          1015.0236 1021.6203
```

According to the Akaike's information criterion and Bayesian one, the best model for this distribuition would be the *log-normal*, since its the lower values.

EU_Sales

EU_Sales is also a continuous variable that represents all the sales of video games in Europe of this top 200 rank.

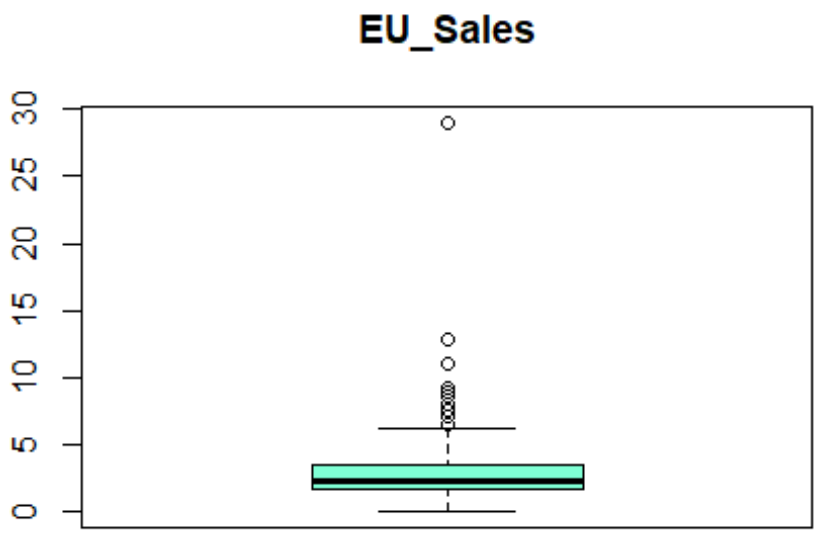
Again we will check some values of the sample using the summary function, and we already have some interesting notations like: there is some video game that didnt have any sale in Europe, since the minimun value found in the database is 0.00. Other thing we can look carefully is the maximun value of sales that is 29.020 milions and its too much below than the maximun value found in the North America sales (41.490 milions).

```
summary(vg_sales$EU_Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   1.705   2.300   3.048   3.522  29.020
```

Just as we did with *NA_Sales*, we can also use a Boxplot to describe graphically the values of *EU_Sales*:

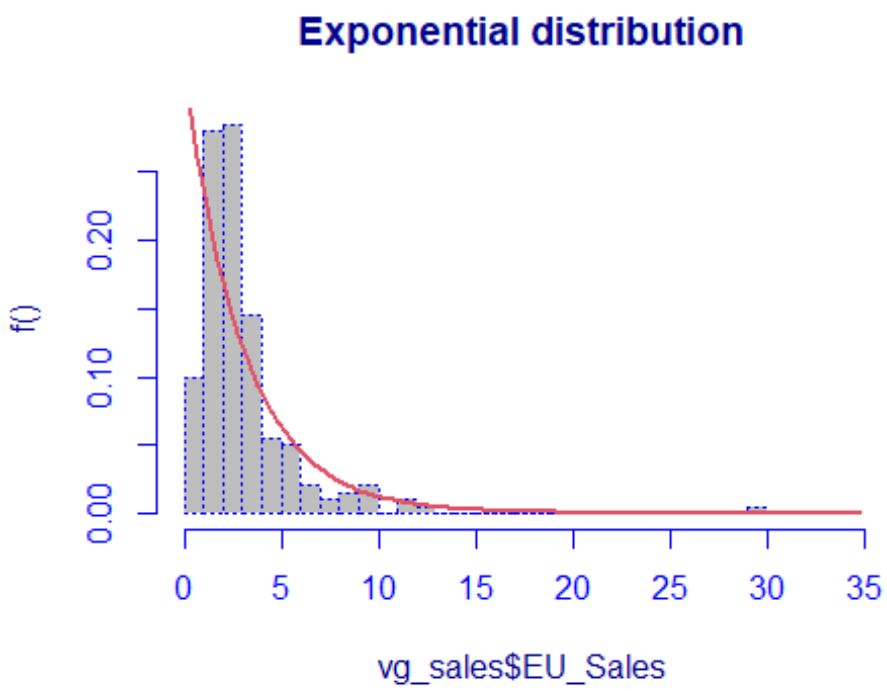
```
library(ggplot2)
boxplot(vg_sales$EU_Sales, main="EU_Sales", col="#7FFFD4")
```



- The bottom segment of the box (the 1st quartil) is 1.705 milions;
- The segment in the middle of the box represents the median value and is 2.300 milions;
- The top segment of the box represents the third quartile and is 3.522 milions;
- The bottom whisker indicates the minimun value of the sample (with no outliers) that is we can see that is close to 0;
- The top whisker indicates the maximun value of the sample (with no outliers) and we can check that is close to 5 milions;
- The sales between nearly 6 milions and 30 milions are all outliers.

Now we can look for the model that fits best for the distribution:

```
library(gamlss)
fit.EXP1 <- histDist(vg_sales$EU_Sales, family=EXP, nbins = 30, main="Exp
onential distribution")
```



Since the *EU_Sales* has values equal to zero, because some video games didn't sell in Europe, I could only do the exponencial distribuition and its the only one suited for this type of cases.

JP_Sales

JP_Sales follows the same description as the others and is also a continuous variable, but this time representing the sales of a single country, Japan.

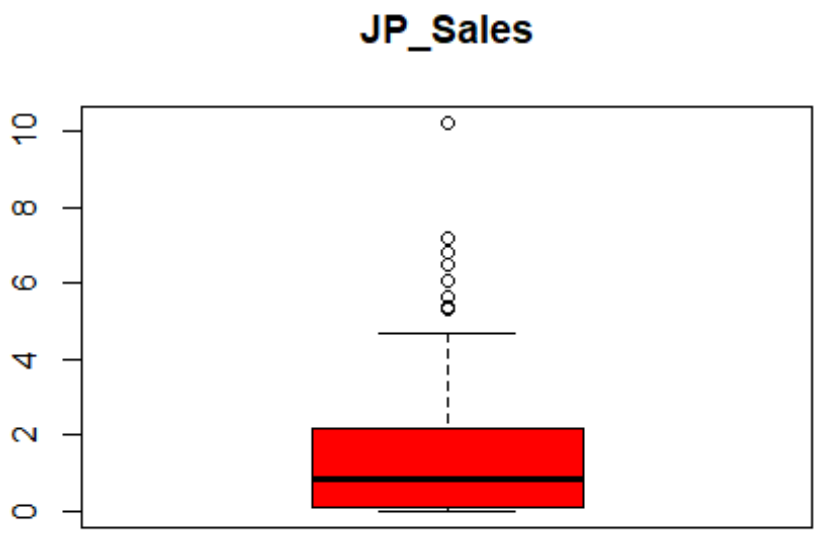
One more time we are going to use the summary function to check the main values of this variable in the database. And as we can check below, in the same way that happened with *EU_Sales* we have a minimum value of sales that is 0.00, that means we have a video game (or more than one) that didn't have considerable sales in Japan. Other curiosity we can have from it is that the maximum value of sales is 10.22 milions, and it is even below to the Europe Sales maximum values that was already low (when compared to the sales in North America).

```
summary(vg_sales$JP_Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1075  0.8300  1.4116  2.1475 10.2200
```

We can also use a Boxplot graph to describe it better:

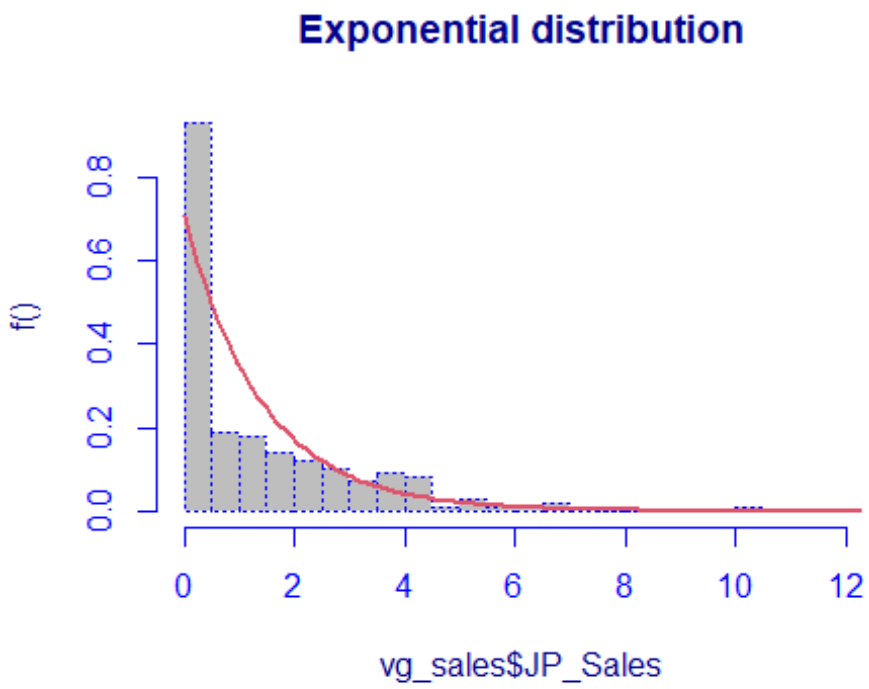
```
library(ggplot2)
boxplot(vg_sales$JP_Sales, main="JP_Sales", col="red")
```



- The bottom segment of the box (the 1st quartil) is 0.1075 milions;
- The the median value is 0.8300 milions;
- The top segment of the box represents the third quartile and is 2.14 milions;
- The bottom whisker indicates the minimun value of the sample (with no outliers) that is we can see that is close to 0;
- The top whisker indicates the maximun value of the sample (with no outliers) and we can check that is close to 5 milions;
- In this case we can see clearly that we have about to 7 outliers that are the sales in the range of nearly 5 milions until 10 milions.

Now we can look for the model that fits best for the distribution:

```
library(gamlss)
fit.EXP2 <- histDist(vg_sales$JP_Sales, family=EXP, nbins = 30, main="Exp
onential distribution")
```



Since the *JP_Sales* has values equal to zero, because some video games didn't sell in Japan, I could only do the exponencial distribuition and its the only one suited for this type of cases.

Other_Sales

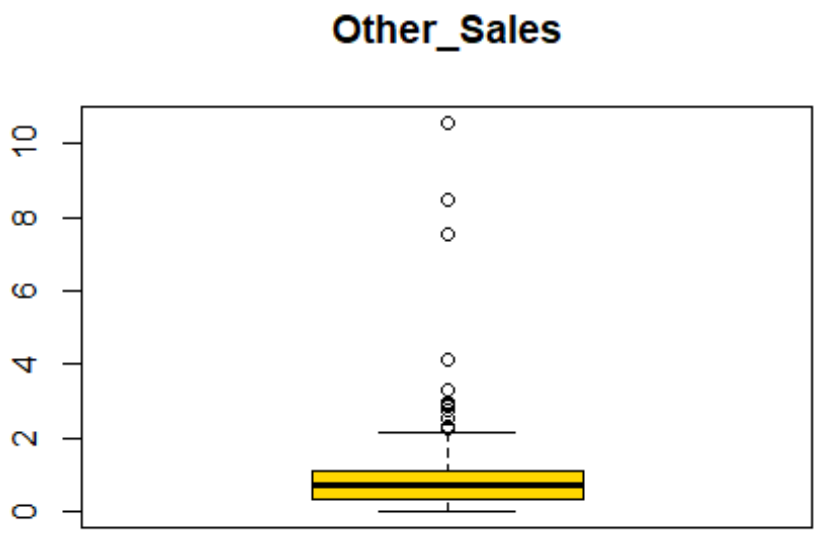
The *Other_Sales* variable represents the sales in the rest of the world of these video games in the top 200 rank. As we can see, the values are very close to the Japan sales, and its below Europe sales and North America sales. Some video games didn't have a sale in the rest of the world (or it was not considerable), the maximun sale of a video game is 10.57 milions and the mean of sale in the rest of the world was 0.93 milions.

```
summary(vg_sales$Other_Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.3300  0.6850  0.9398  1.0750 10.5700
```

Lets use a boxplot to describe it better:

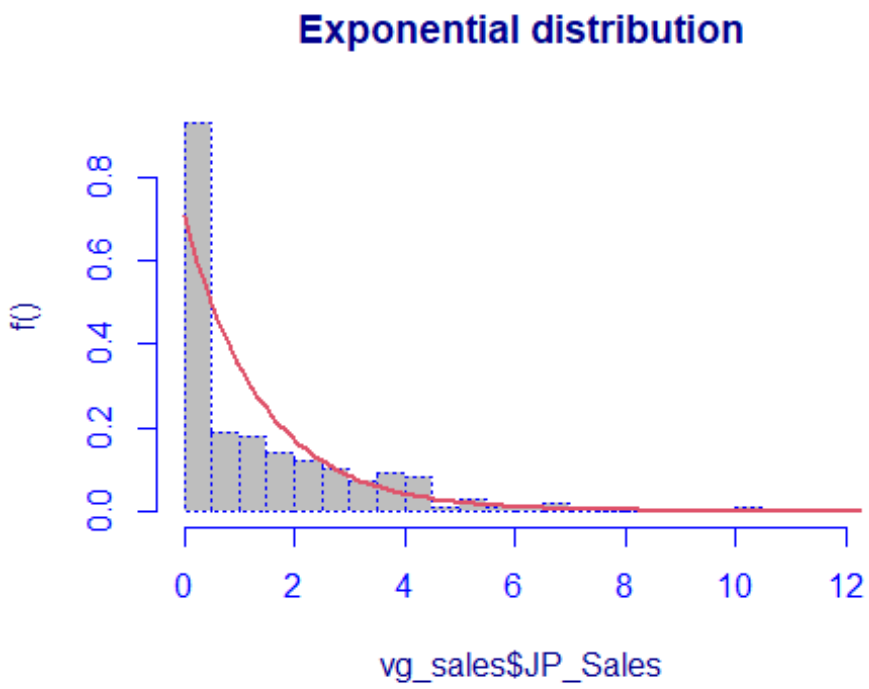
```
library(ggplot2)
boxplot(vg_sales$Other_Sales, main="Other_Sales", col="#FFD700")
```



- The bottom segment of the box (the 1st quartil) is 0.33 milions;
- The the median value is 0.68 milions;
- The top segment of the box represents the third quartile and is 1.075 milions;
- The bottom whisker indicates the minimun value of the sample (with no outliers) that is we can see that is close to 0;
- The top whisker indicates the maximun value of the sample (with no outliers) and we can check that is close to 2 milions;
- The outliers are in the range of 3 milions until more than 10 milions of sales.

Now we can look for the model that fits best for the distribution:

```
library(gamlss)
fit.EXP3 <- histDist(vg_sales$JP_Sales, family=EXP, nbins = 30, main="Exp
onential distribution")
```



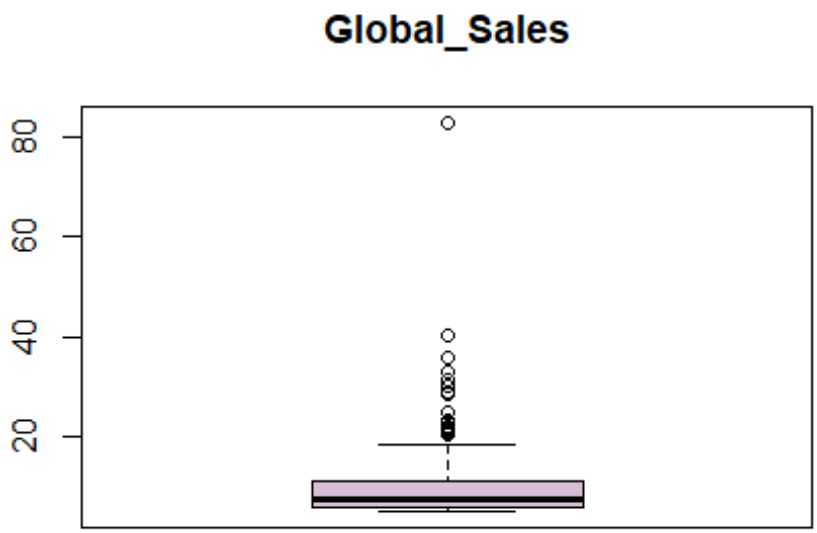
Other_Sales also has values equal to zero, because some video games didn't sell in other places, I could only do the exponencial distribuition and its the only one suited for this type of cases.

Global_Sales

The *Global_Sales* variable is the sum of all the variables mentioned and represents the total sale of each video game worldwide, so as we can presume it has the largest numbers in all statistical representations.

When we use the summary function, its possible to see that the minimun sale of a video game is 5.08 milions and the maximun is 82.740 milions, and it both represents the sales of the last one in the rank and the first one in the rank, since our database is ordered by the Global Sales.

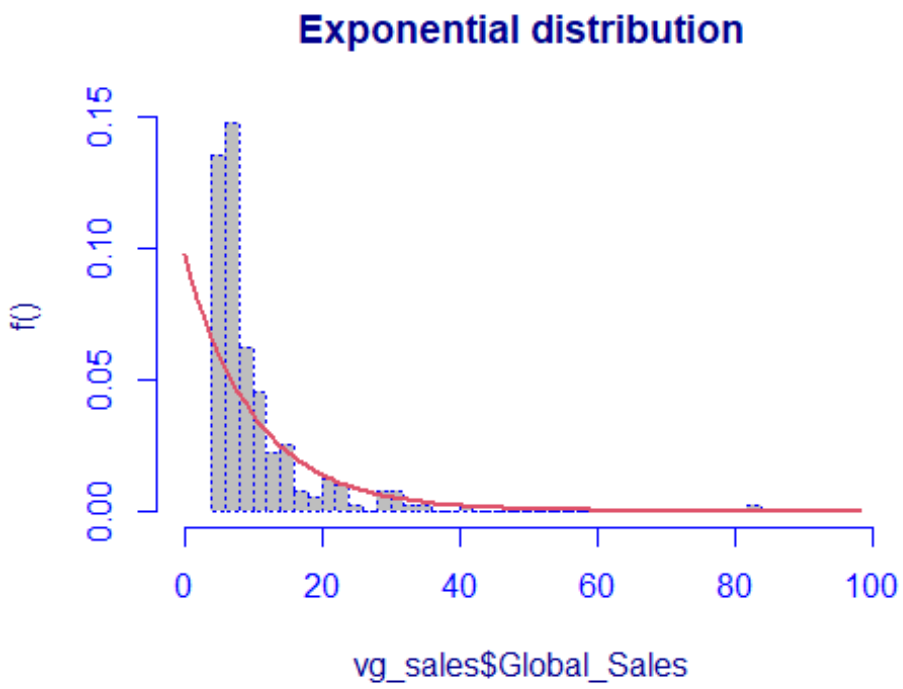
```
summary(vg_sales$Global_Sales)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   5.080   5.838   7.325  10.291  11.217  82.740

library(ggplot2)
boxplot(vg_sales$Global_Sales, main="Global_Sales", col="#D8BFD8")
```



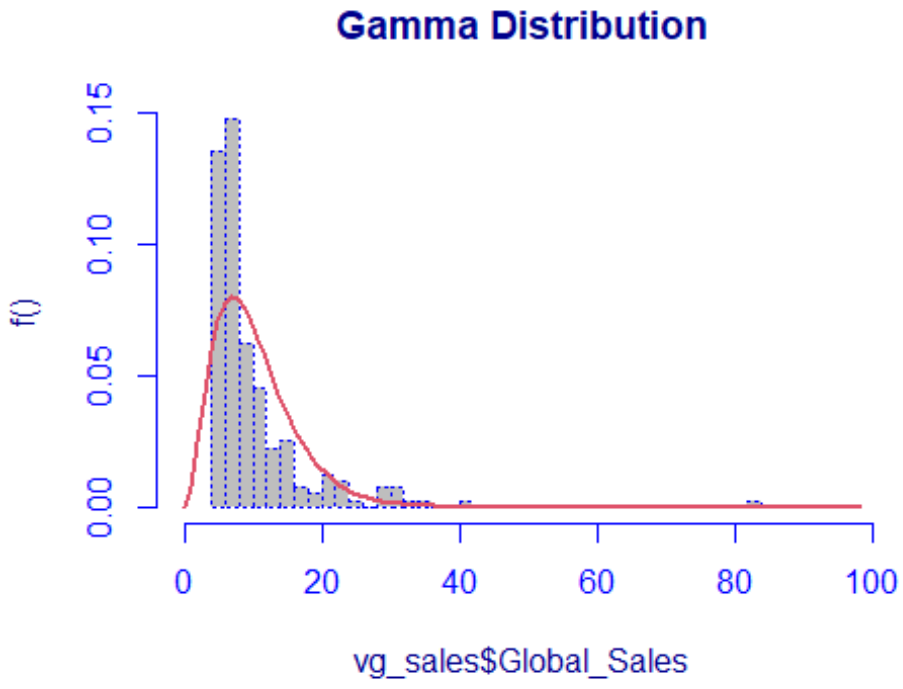
- The bottom segment of the box (the 1st quartil) is 5.83 milions;
- The the median value is 7.32 milions;
- The top segment of the box represents the third quartile and is 11.217 milions;
- The bottom whisker indicates the minimun value of the sample (with no outliers) that it must be close to 5 milions.
- The top whisker indicates the maximun value of the sample (with no outliers) and we can check that it must be close to 18 milions.
- As we can see, in this case we have the largest range of outliers, that goes since 20 milions until more than 80 milions sales.

Now we can look for the model that fits best for the distribution:

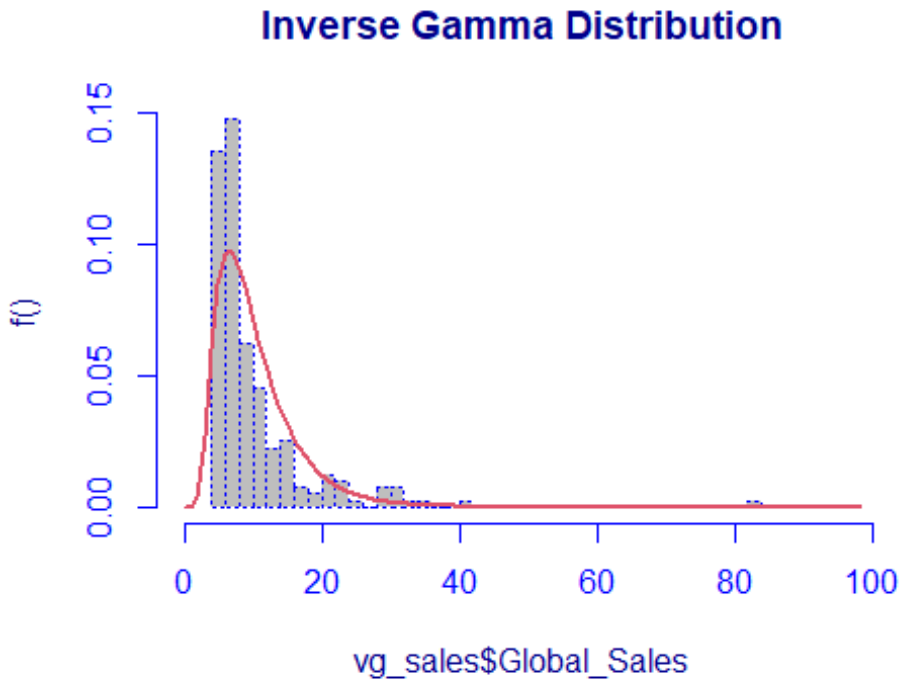
```
library(gamlss)
fit.EXP4 <- histDist(vg_sales$Global_Sales, family=EXP, nbins = 30, main=
"Exponential distribution")
```



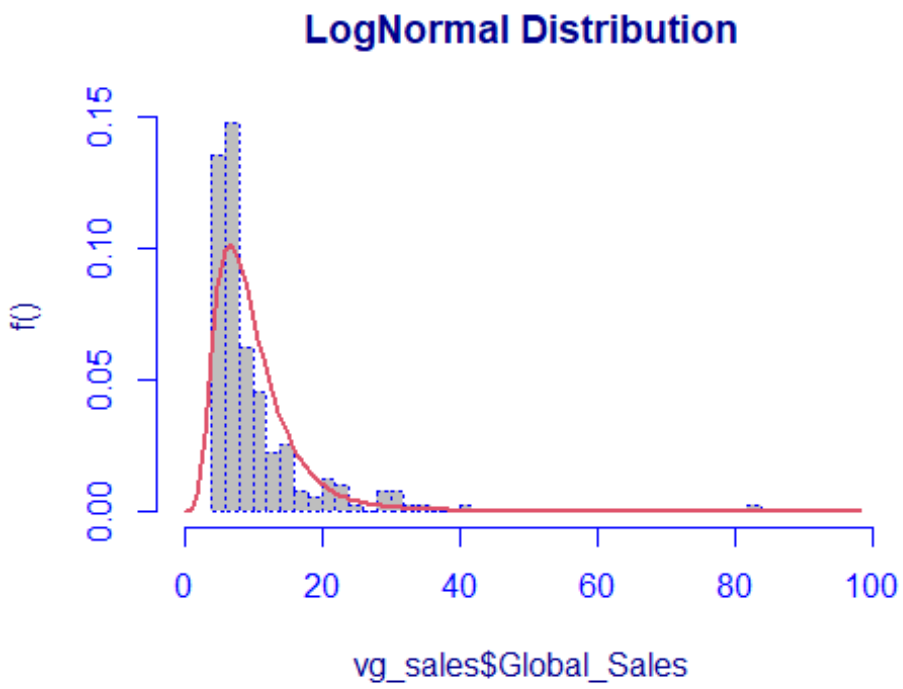

```
fit.GA4 <- histDist(vg_sales$Global_Sales, family=GA, nbins=30, main = "G
amma Distribution")
```



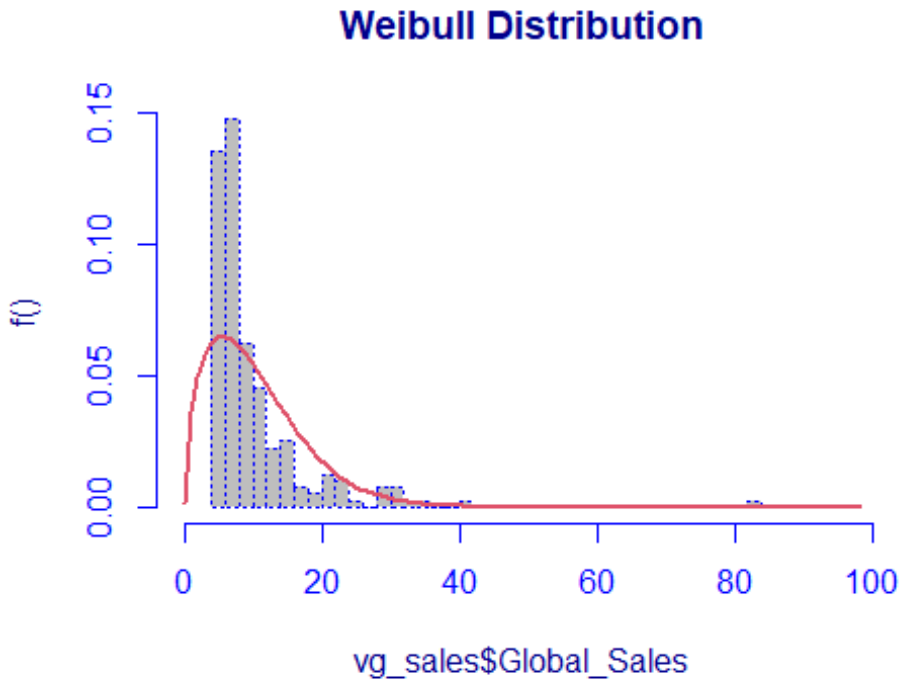
```
fit.IG4 <- histDist(vg_sales$Global_Sales, family=IG, nbins=30, main = "I
nverse Gamma Distribution")
```




```
fit.LOGNO4 <- histDist(vg_sales$Global_Sales, family=LOGNO, nbins=30, mai
n = "LogNormal Distribution")
```




```
fit.WEI4 <- histDist(vg_sales$Global_Sales,family=WEI, nbins=30, main = "
Weibull Distribution")
```



```
data.frame(row.names = c("Exponential", "Gamma", "Inverse Gaussian", "Log
-Normal", "Weibull"),
AIC=c(AIC(fit.EXP4), AIC(fit.GA4), AIC(fit.IG4), AIC(fit.LOGNO4), AIC(fit
.WEI4)),
SBC=c(fit.EXP4$sbc, fit.GA4$sbc, fit.IG4$sbc, fit.LOGNO4$sbc, fit.WEI4$sb
c))
```

```
##                       AIC      SBC
## Exponential      1334.490 1337.789
## Gamma            1229.066 1235.662
## Inverse Gaussian 1173.150 1179.746
## Log-Normal       1171.760 1178.357
## Weibull          1276.019 1282.615
```

According to the Akaike's information criterion and Bayesian one, the best model for this distribuition would be the *log-normal*.

Multivariate Analysis

Now lets do the multivariate analysis which involves observation and analysis of more than one statistical outcome variable at a time.

Principal Component

Principal Component Analysis or PCA is a multivariate analysis technique that can be used to analyze interrelationships between a large number of variables and explain these variables in terms of their inherent dimensions (Components).

The goal is to find a way to condense the information contained in several original variables into a smaller set of statistical variables (components) with a minimal loss of information.

The number of main components becomes the number of variables considered in the analysis, but generally the first components are the most important since they explain most of the total variation.

The main components are usually extracted via the covariance matrix, but they can also be extracted via the correlation matrix.

First of all we cannot use PCA to analyze categorical variables, so we will delete these columns from this and analyze only the numerical variables.

I will change the variable *Name* into row names to have a more clear view of graphs and correlation, but before doing tht I realized that many video games have the same name and its only difference were the platform, so I will bring into like a rowname doing a combination of *Name* with *Platform* to have a unique value. I also decided to delete the variable *Year* and working only with the sales. We can see all this work below:

```
vg_sales.pca <- vg_sales[1:200, 6:10]

for (i in 1:nrow(vg_sales)){
  rownames(vg_sales.pca)[i]<- paste(vg_sales[i,1], levels(vg_sales$Platfo
rm)[as.numeric(vg_sales[i,2])], sep = " - ")
}

head(vg_sales.pca)

## # A tibble: 6 x 5
##   NA_Sales EU_Sales JP_Sales Other_Sales Global_Sales
##      <dbl>    <dbl>    <dbl>       <dbl>        <dbl>
## 1     41.5    29.0      3.77       8.46          82.7
## 2     29.1     3.58     6.81       0.77          40.2
## 3     15.8    12.9      3.79       3.31          35.8
## 4     15.8    11.0      3.28       2.96          33
```

```
## 5     11.3     8.89    10.2        1             31.4
## 6     23.2     2.26     4.22       0.580         30.3
```

Now lets check the variance of each variable to see how far each value is from the central (average) value.

```
apply (vg_sales.pca, 2, var)
```

```
##     NA_Sales     EU_Sales     JP_Sales  Other_Sales Global_Sales
##    22.349980     7.926434     2.935649     1.428058    68.200161
```

Now lets standardize these values before aplying PCA because of the high differences between the variables, and if we dont standardize the PCA can be influenced by outliers.

```
scaled_vg_sales.pca <- apply(vg_sales.pca, 2, scale)
head(scaled_vg_sales.pca)
```

```
##      NA_Sales   EU_Sales JP_Sales Other_Sales Global_Sales
## [1,] 7.741472  9.2249668 1.376495  6.29294036     8.772884
## [2,] 5.116448  0.1889257 3.150772 -0.14213226     3.626571
## [3,] 2.317974  3.4921954 1.388168  1.98336637     3.091354
## [4,] 2.296822  2.8279896 1.090509  1.69048270     2.749881
## [5,] 1.349190  2.0749862 5.140998  0.05033415     2.552505
## [6,] 3.872682 -0.2799255 1.639135 -0.30112625     2.418095
```

To calculate the Principal Component, we will use the *cov()* function to calculate the covariance matrix and then use the *eigen()* function to calculate the eigenvectors and eigenvalues.

The *eigen* function results in an object containing the ordered eigenvalues and the corresponding eigenvectors of the matrix.

```
std_sales.cov <- cov(scaled_vg_sales.pca)
sales_eigen <- eigen(std_sales.cov)
str(sales_eigen)
```

```
## List of 2
##  $ values : num [1:5] 3.19 9.73e-01 4.97e-01 3.42e-01 4.00e-07
##  $ vectors: num [1:5, 1:5] 0.485 0.485 0.278 0.378 0.556 ...
##  - attr(*, "class")= chr "eigen"
```

Just as an example, we take the first two sets of loadings and store them in the matrix called eg.

```
(eg <- sales_eigen$vectors[,1:2])
```

```
##           [,1]        [,2]
## [1,] 0.4852387  0.04726992
## [2,] 0.4854150 -0.14353328
## [3,] 0.2782807  0.80816670
```

```
## [4,] 0.3777546 -0.56563800
## [5,] 0.5556771  0.06390614
```

Now we are going to place the points of eigenvectors in a positive direction to have a more logical interpretation of the graphs. The set of loadings for the first principal component (PC1) and second principal component (PC2) are shown below:

```
eg <- -eg
row.names(eg) <- c("NA_Sales", "EU_Sales", "JP_Sales", "Other_Sales", "Gl
obal_Sales")
colnames(eg) <- c("PC1", "PC2")
eg
```

```
##                     PC1         PC2
## NA_Sales     -0.4852387 -0.04726992
## EU_Sales     -0.4854150  0.14353328
## JP_Sales     -0.2782807 -0.80816670
## Other_Sales  -0.3777546  0.56563800
## Global_Sales -0.5556771 -0.06390614
```

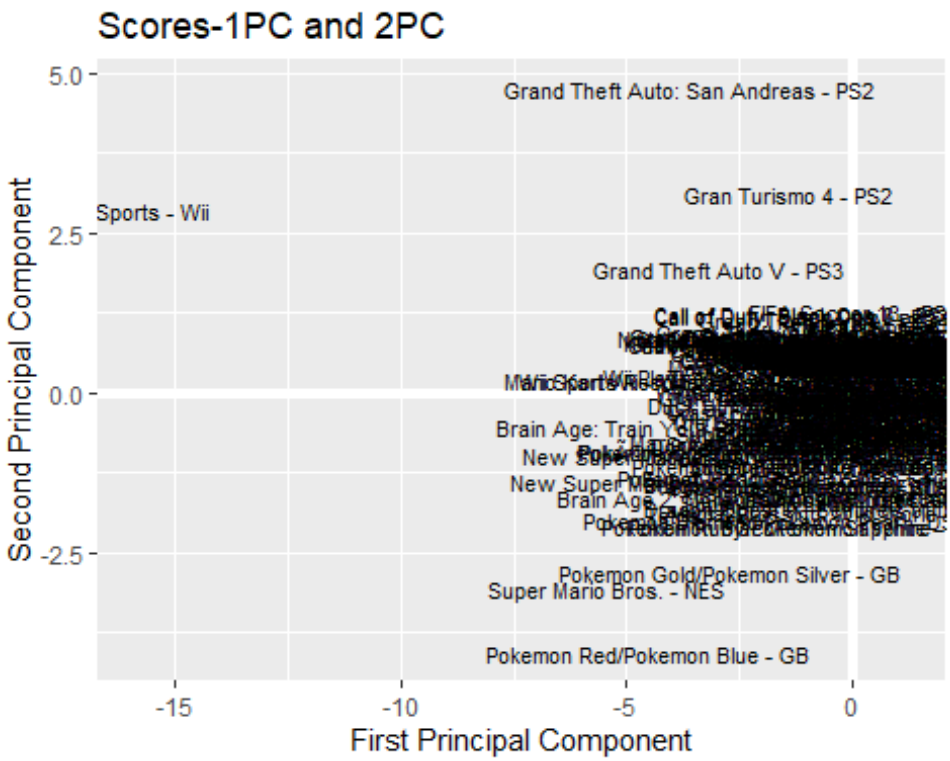
Now lets project *n* datapoints into the first eigenvector, the projected values are the principal component scores of each observation:

```
PC1 <- scaled_vg_sales.pca %*%  eg[,1]
PC2 <- scaled_vg_sales.pca %*%  eg[,2]

PC <- data.frame(Video_Games = row.names(vg_sales.pca), PC1, PC2)
head(PC, digits = 3)
```

```
##                     Video_Games        PC1        PC2
## 1              Wii Sports - Wii -15.869528  2.8445988
## 2       Super Mario Bros. - NES  -5.412716 -3.0732416
## 3          Mario Kart Wii - Wii  -5.673255  0.1941158
## 4       Wii Sports Resort - Wii  -4.957356  0.1964937
## 5 Pokemon Red/Pokemon Blue - GB  -4.529932 -4.0553794
## 6                   Tetris - GB  -3.429363 -1.8727935
```

We've calculated the 1st and 2nd principal component of each video game, and now we can plot a two dimensional view of the data:

```
library(ggplot2)
library(modelr)

ggplot(PC, aes(PC1, PC2)) +
modelr::geom_ref_line(h = 0) +
modelr::geom_ref_line(v = 0) +
geom_text(aes(label = Video_Games), size = 3) +
xlab("First Principal Component") +
ylab("Second Principal Component") +
ggtitle("Scores-1PC and 2PC")
```



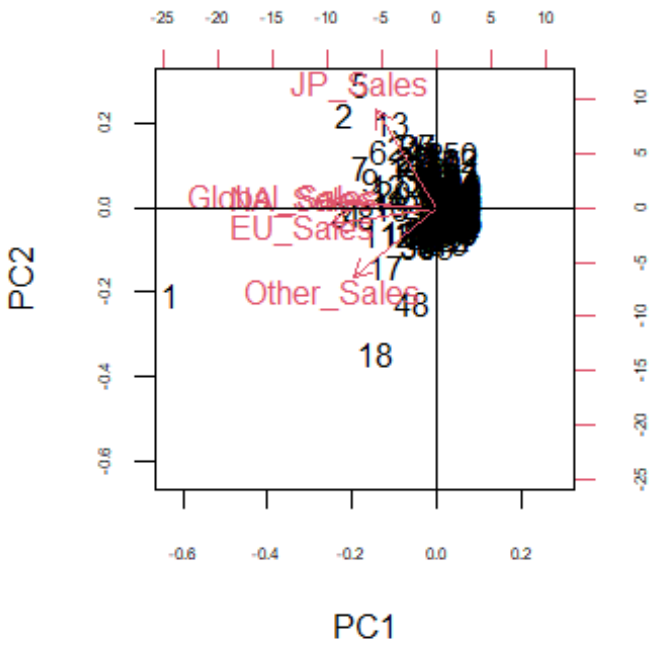
The first principal component (x-axis) seems to correspond to the sales of North America, Europe and the Global, which are the higher sales.

The second component (y-axis) is strongly represented by Japan and Other sales.

We can also see that the majority of video games are close to the origin, which means they are close to average in both categories.

And in these graphs we can also notice 3 or 4 outliers in the group.

```
set.seed(123)
res.pca <- prcomp(scaled_vg_sales.pca, center = TRUE, scale. = FALSE)
biplot(res.pca, cex.axis = 0.5, scale=1)
abline(h=0)
abline(v=0)
```



In the graph above the angle between the arrows give us information about the correlation of two variables. Based on the concepts below:

- angle close to 0 → correlation close to 1.
- angle close to 90 ◦ → correlation close to 0.
- angle close to 180 ◦ → correlation close to -1.

In this case the angles between *Global_Sales* and *NA_Sales* is almost zero, we can see that because they overlap each other, it means that the correlation is close to 1, which means they are highly correlated.

In this case we also notice that the video game the occupies the first place in the rank is a big outlier in the plot.

(Cumulative) 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.

```
PVE <- sales_eigen$values/sum(sales_eigen$values)
round(PVE,3)

## [1] 0.638 0.195 0.099 0.068 0.000
```

- The first principal component explains 63,8% of the variability;
- The second principal component explains 19,5% of the variability;
- The first and second component together explains 83,3% of the variability.

According to this approach, the first principal components that explain at least 80% of the total variance are retained. In this case the first principal component plus the second principal component explains 83,3%, so they must be retained.

Kaiser Rule

You can use the eigenvalue size to determine the number of major components. Retain the main components with the largest eigenvalues. For example, using the *Kaiser* rule, you use only the main components with eigenvalues that are greater than 1.

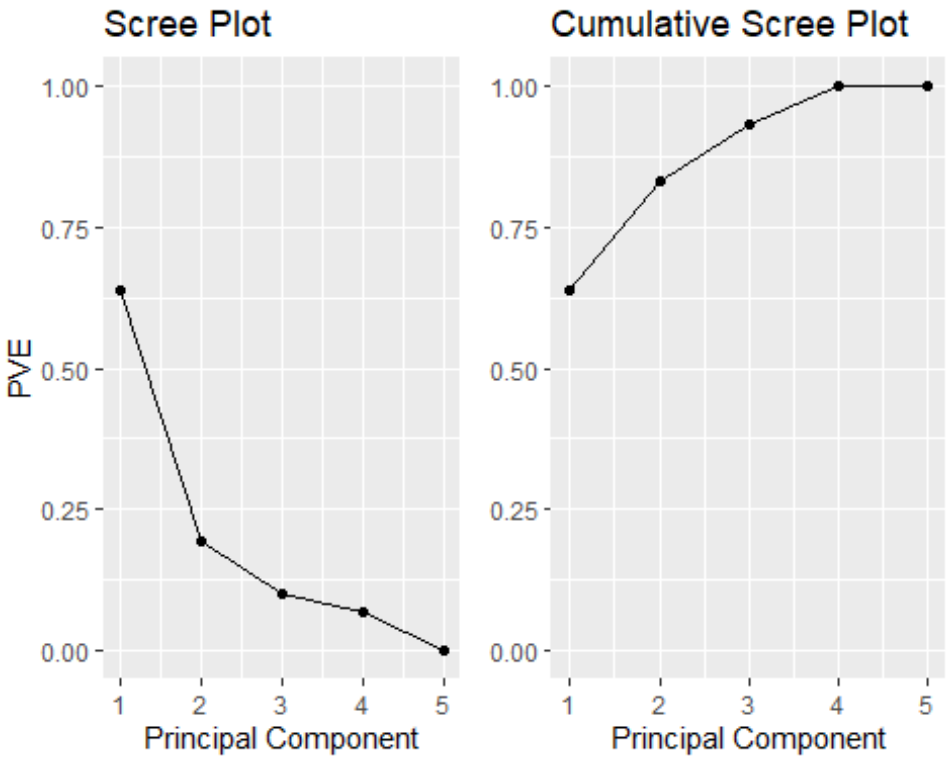
```
round(sales_eigen$values,3)

## [1] 3.188 0.973 0.497 0.342 0.000
```

Based on this, we will use only the first PC.

Scree Plot

To visually compare the size of the eigenvalues, use the scree graph. The scree graph can help you determine the number of components based on the size of the eigenvalues.

```
PVEplot <- qplot(c(1:5), PVE) +
  geom_line()+
  xlab("Principal Component")+
  ylab("PVE")+
  ggtitle ("Scree Plot")+
  ylim(0,1)

cumPVE <- qplot(c(1:5), cumsum(PVE)) +
  geom_line()+
  xlab("Principal Component") +
  ylab(NULL)+
  ggtitle("Cumulative Scree Plot") +
  ylim(0,1)

library(gridExtra)

grid.arrange(PVEplot, cumPVE, ncol=2)
```



To determine the number of components, the scree plot suggest to select the number of PC corresponding to the value of the elbow where the curve becomes flat or the elbow where the PVE significantly drops off.

In this example the result is not so clear just by looking to the graphs, but according to what we may see, its reasonable to mantain the first 2 PC.

Final result of PCA

Based on these different methods we had different results. Based on the cumulative proportion of the variance, we should retain the first 2 principal components. Based on the Kaiser Rule, we should retain only the first PC and based on the scree plot we should retain 2 PCs.

The Scree Plot has the issue to be the most subjective method of analysis, cause sometimes the elbow is not so clear to see. So I decided to use the cumulative proportion of variance explained, cause it will retain more PCs when compared to the Kaiser Rule.

Cluster Analysis

Clustering is the automatic grouping of similar instances, an unsupervised classification of data. That is, an algorithm that clusters data classifies them into data sets that 'resemble' in some way - regardless of predefined classes. The groups generated by this classification are called clusters.

The difference between clustering and PCA is that the PCA looks to find a low-dimensional representation of the observations that explain a good fraction of the variance while the clustering looks to find homogeneous subgroups among the observations.

In the clustering we are interested in the units of a group (rows of a dataset) and the classification of observations into groups requires some methods for computing the distance or the dissimilarity between each pair of observations.

The choice of dissimilarity or distance measures is a critical step in clustering. It defines how the similarity of two elements is calculated and it will influence the shape of the clusters.

To start our clutering analysis, lets first standardize de datas again, first taking out the categorical values one more time:

```
vg_sales.cl <- vg_sales[1:200, 6:10]
vg_sales.scaled <- apply(vg_sales.cl, 2, scale)
head(vg_sales.scaled)
```

```
##      NA_Sales   EU_Sales JP_Sales Other_Sales Global_Sales
## [1,] 7.741472  9.2249668 1.376495  6.29294036     8.772884
## [2,] 5.116448  0.1889257 3.150772 -0.14213226     3.626571
## [3,] 2.317974  3.4921954 1.388168  1.98336637     3.091354
## [4,] 2.296822  2.8279896 1.090509  1.69048270     2.749881
## [5,] 1.349190  2.0749862 5.140998  0.05033415     2.552505
## [6,] 3.872682 -0.2799255 1.639135 -0.30112625     2.418095
```

Euclidean Distance

The euclidean distance is a distance measure between a pair of samples *p* and *q* in an n-dimensional feature space. We will subset the first 10 columns and rows and round the result to have a better visualization:

```
dist.eucl <- dist(vg_sales.scaled, method = "euclidean")
round(as.matrix(dist.eucl)[1:10, 1:10], 2)
```

```
##        1     2     3     4     5     6     7     8     9    10
## 1   0.00 12.63 10.64 11.32 13.56 13.76 12.45 12.16 12.78 14.26
## 2  12.63  0.00  5.16  4.83  4.79  2.35  4.77  4.91  4.00  4.24
## 3  10.64  5.16  0.00  0.86  4.59  4.73  2.39  1.71  2.48  5.92
## 4  11.32  4.83  0.86  0.00  4.54  4.06  2.22  0.91  1.83  5.19
## 5  13.56  4.79  4.59  4.54  0.00  4.93  2.70  4.58  3.54  7.32
```

```
## 6  13.76  2.35  4.73  4.06  4.93  0.00  4.23  3.75  2.88  2.51
## 7  12.45  4.77  2.39  2.22  2.70  4.23  0.00  2.16  1.57  6.12
## 8  12.16  4.91  1.71  0.91  4.58  3.75  2.16  0.00  1.38  4.80
## 9  12.78  4.00  2.48  1.83  3.54  2.88  1.57  1.38  0.00  4.57
## 10 14.26  4.24  5.92  5.19  7.32  2.51  6.12  4.80  4.57  0.00
```

In this symmetric matrix, each value represents the distance between units, each units as a position on the rank. The values on the diagonal represent the distance between units and themselves (which is zero).

But lets compute the euclidean distance for 200 observations, just to save it in a variable for future analysis:

```
dist.eucl.200 <- dist(vg_sales.scaled, method = "euclidean")
```

Manhattan Distance

The Manhattan distance captures the distance between two points by aggregating the pairwise absolute difference between each variable.

```
dist.man <- dist(vg_sales.scaled, method = "manhattan")
round(as.matrix(dist.man)[1:10, 1:10], 2)
```

```
##        1     2     3     4     5     6     7     8     9    10
## 1   0.00 25.02 21.16 22.75 29.77 26.59 26.03 24.54 25.77 28.48
## 2  25.02  0.00 10.53 10.23  8.91  4.59  8.95 10.54  8.19  7.01
## 3  21.16 10.53  0.00  1.62  8.61  8.54  4.87  3.40  4.62 12.03
## 4  22.75 10.23  1.62  0.00  7.59  7.56  3.85  1.78  3.59 10.45
## 5  29.77  8.91  8.61  7.59  0.00  8.87  4.07  6.78  5.96 12.86
## 6  26.59  4.59  8.54  7.56  8.87  0.00  8.28  7.21  5.41  4.00
## 7  26.03  8.95  4.87  3.85  4.07  8.28  0.00  2.82  3.20 12.21
## 8  24.54 10.54  3.40  1.78  6.78  7.21  2.82  0.00  2.45  9.40
## 9  25.77  8.19  4.62  3.59  5.96  5.41  3.20  2.45  0.00  9.01
## 10 28.48  7.01 12.03 10.45 12.86  4.00 12.21  9.40  9.01  0.00
```

Similar to the euclidean distance matrix, in this case each value represents the distance between units, each units as a position on the rank. The values on the diagonal represent the distance between units and themselves (which is zero).

Now lets compute the manhattan distance for 200 observations and save it in a variable just to save it for future analysis:

```
dist.man.200 <- dist(vg_sales.scaled, method = "manhattan")
```

Computing Distance for Mixed Data

The original data set contains mixed data (character, factors and numeric) and we have a solution to calculate de distance of mixed data that its to use the gower distance, applying the *daisy()* function that computes de distance of datasets with factors and numerical values. In this case I have to take the *name* column cause it is a character type variable.

```
library(cluster)

gower.dist <- daisy(vg_sales [,-1])
round(as.matrix(gower.dist)[1:10,1:10],2)

##       1    2    3    4    5    6    7    8    9   10
## 1  0.00 0.60 0.37 0.28 0.64 0.59 0.54 0.41 0.43 0.64
## 2  0.60 0.00 0.44 0.44 0.38 0.30 0.29 0.43 0.30 0.22
## 3  0.37 0.44 0.00 0.14 0.39 0.39 0.30 0.16 0.17 0.46
## 4  0.28 0.44 0.14 0.00 0.38 0.38 0.29 0.14 0.16 0.44
## 5  0.64 0.38 0.39 0.38 0.00 0.26 0.32 0.37 0.36 0.45
## 6  0.59 0.30 0.39 0.38 0.26 0.00 0.39 0.37 0.36 0.30
## 7  0.54 0.29 0.30 0.29 0.32 0.39 0.00 0.27 0.17 0.47
## 8  0.41 0.43 0.16 0.14 0.37 0.37 0.27 0.00 0.16 0.42
## 9  0.43 0.30 0.17 0.16 0.36 0.36 0.17 0.16 0.00 0.43
## 10 0.64 0.22 0.46 0.44 0.45 0.30 0.47 0.42 0.43 0.00
```

Looking at this we can check, for example, that 1(*Wii Sports*) and 10 (*Duck Hunt*) have low similarities while 3 (*Mario Kart Wii*) and 4 (*Wii Sports Resort*) have high similarities.

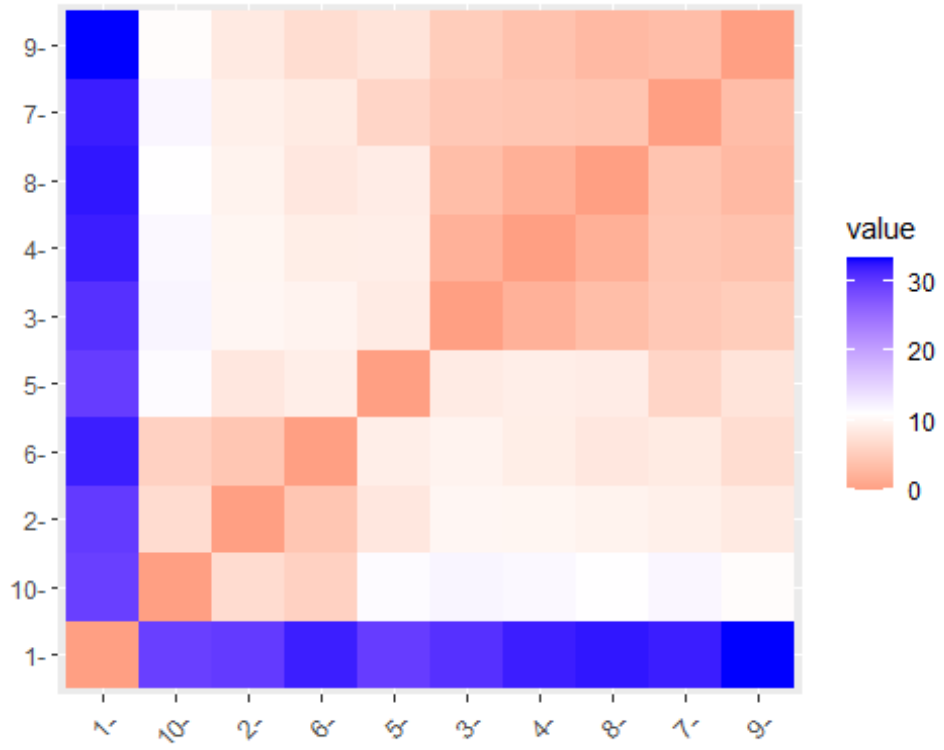
Now lets do it for the 200 observations and save it into a variable:

```
gower.dist.200 <- daisy(vg_sales [,-1])
```

Distance Matrices

We can do the same observations by looking at the graphical visualization of the distance matrix. I will use only 10 observations to have a more clear view:

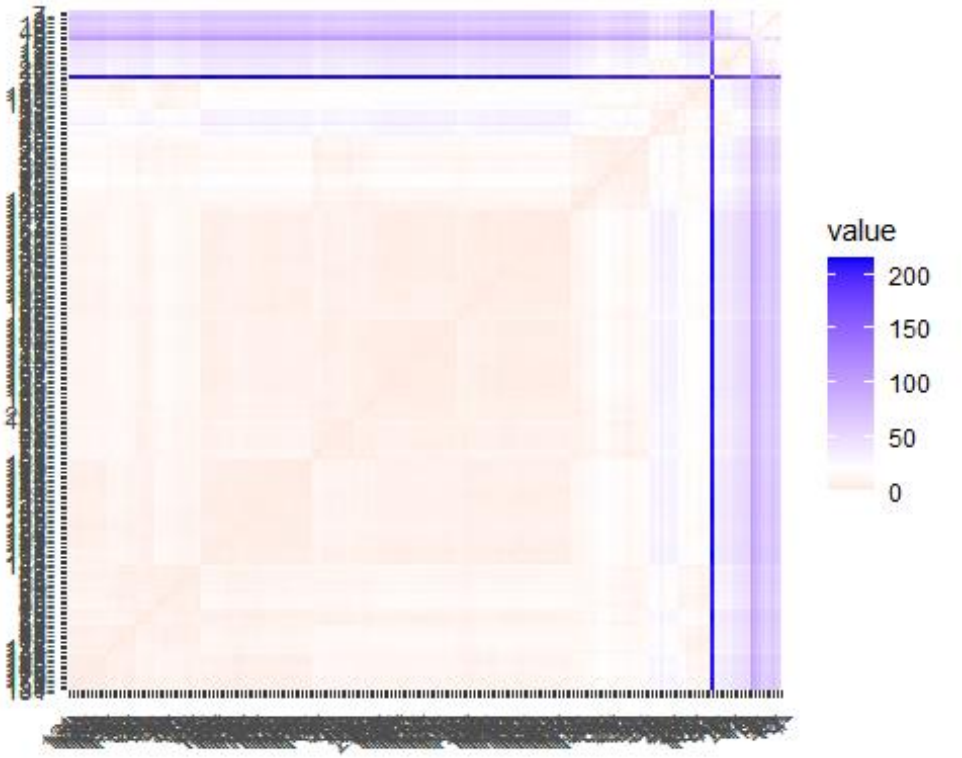
```
library(factoextra)

dist.eucl.1 <- as.matrix(dist.eucl)
dist.eucl.1 <- dist.eucl.1[1:10,1:10]
dist.eucl.1 <- dist(dist.eucl.1, method = "euclidean")
fviz_dist(dist.eucl.1)
```



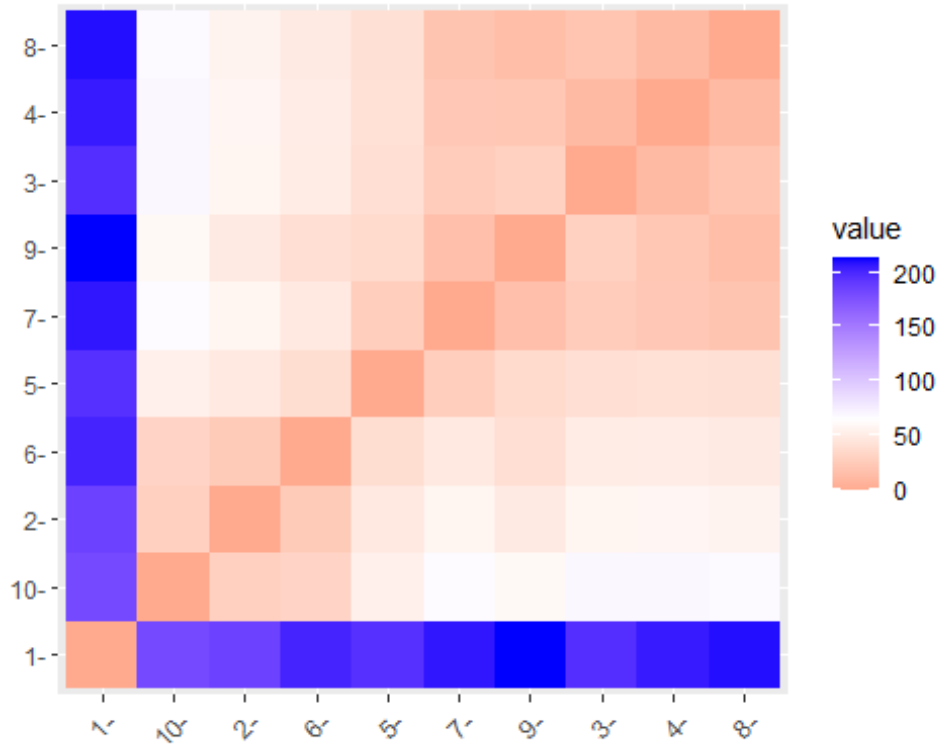
The color level is proportional to the value of the dissimilarity between observations. Red indicates high similarity while blue indicates low similarity, in this case based on the euclidean distance. So with that we can say that number 9 (*New Super Mario Bros. Wii*) and number 1 (*Wii Sports*) in this rank have low similarities while 4 (*Wii Sports Resort*) and 3 (*Mario Kart Wii*) have high similarities.

The graph for 200 observations:

```
library(factoextra)
dist.eucl.2 <- as.matrix(dist.eucl.200)
dist.eucl.2 <- dist.eucl.2
dist.eucl.2 <- dist(dist.eucl.2, method = "euclidean")
fviz_dist(dist.eucl.2)
```



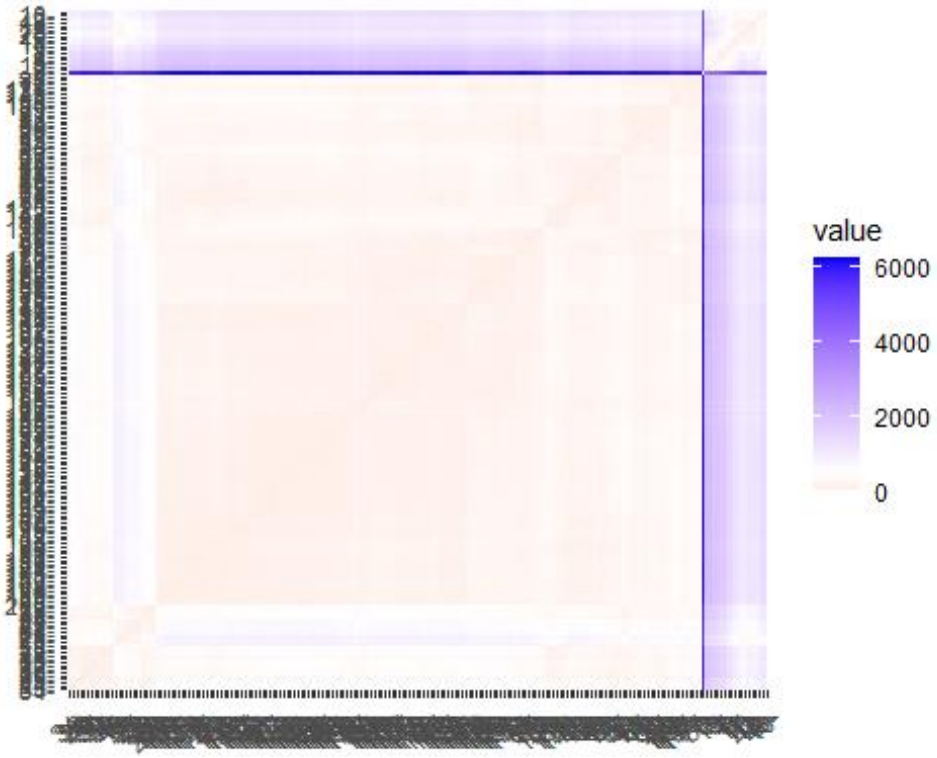

```
library(factoextra)
dist.man.1 <- as.matrix(dist.man)
dist.man.1 <- dist.man.1[1:10,1:10]
dist.man.1 <- dist(dist.man.1, method = "manhattan")
fviz_dist(dist.man.1)
```



This other graph was built usind the manhattan distance and has the same kind of interpretation, blue means less similarities while red means high similarities. In this case, for example, 8 (*Wii Play*) and 1 (*Wii Sports Resort*) have low similarities, while 8 (*Wii Play*) and 4 (*Wii Sports Resort*) have high similarities.

And the graph for the 200 observations:

```
library(factoextra)
dist.man.2 <- as.matrix(dist.man)
dist.man.2 <- dist.man.2
dist.man.2 <- dist(dist.man.2, method = "manhattan")
fviz_dist(dist.man.2)
```


value
6000
4000
2000
0

Hierarchical Clustering

Hierarchical clustering is a method which seeks to build a hierarchy of clusters and this method do not require the number of clusters *K* as an input.

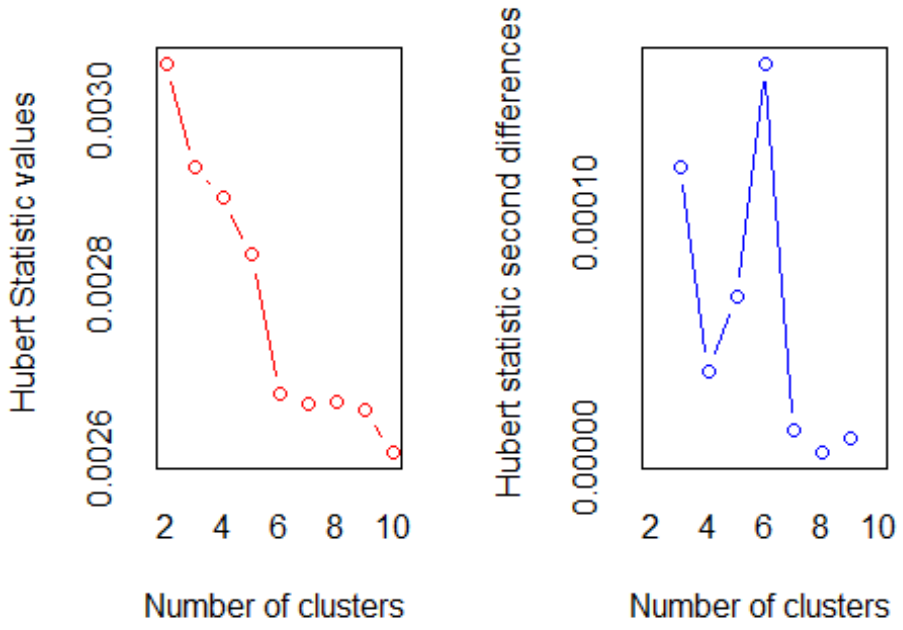
Strategies for hierarchical clustering generally fall into two types:

- Agglomerative -> This is a "bottom-up" approach: each observation starts in its own cluster (leaf), and pairs of clusters are merged as one moves up the hierarchy. This process goes on until there is just one single big cluster (root).
- Divisive -> This is a "top-down" approach: all observations start in one cluster (root), and splits are performed recursively as one moves down the hierarchy. This process goes on until all observations are in their own cluster (leaf).

Average Linkage Method and Euclidean Distance

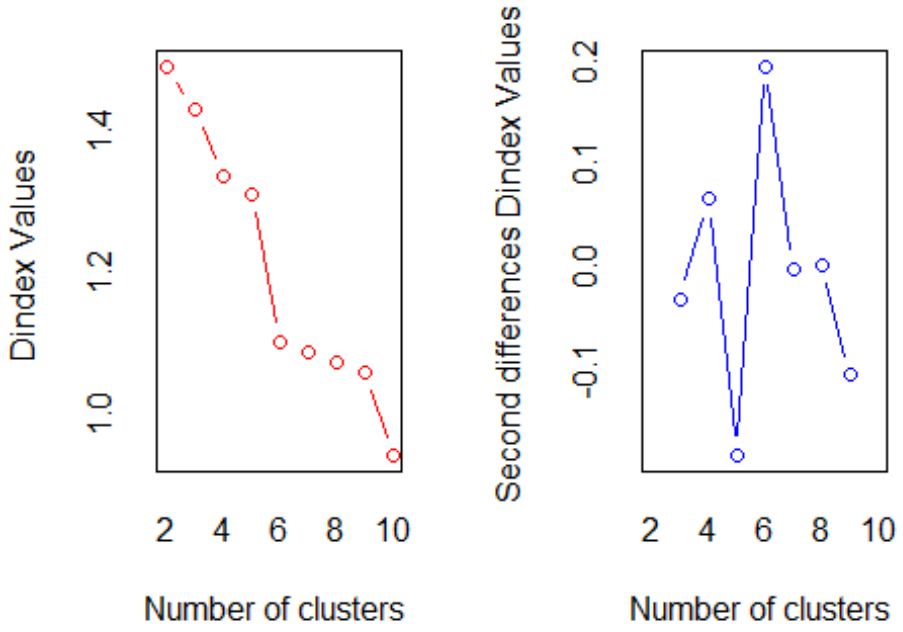
```
library(NbClust)
res.alm <- NbClust(vg_sales.scaled, distance = "euclidean", min.nc = 2, m
ax.nc = 10,
method = "average")
```

```
## [1] "Frey index : No clustering structure in this data set"
```



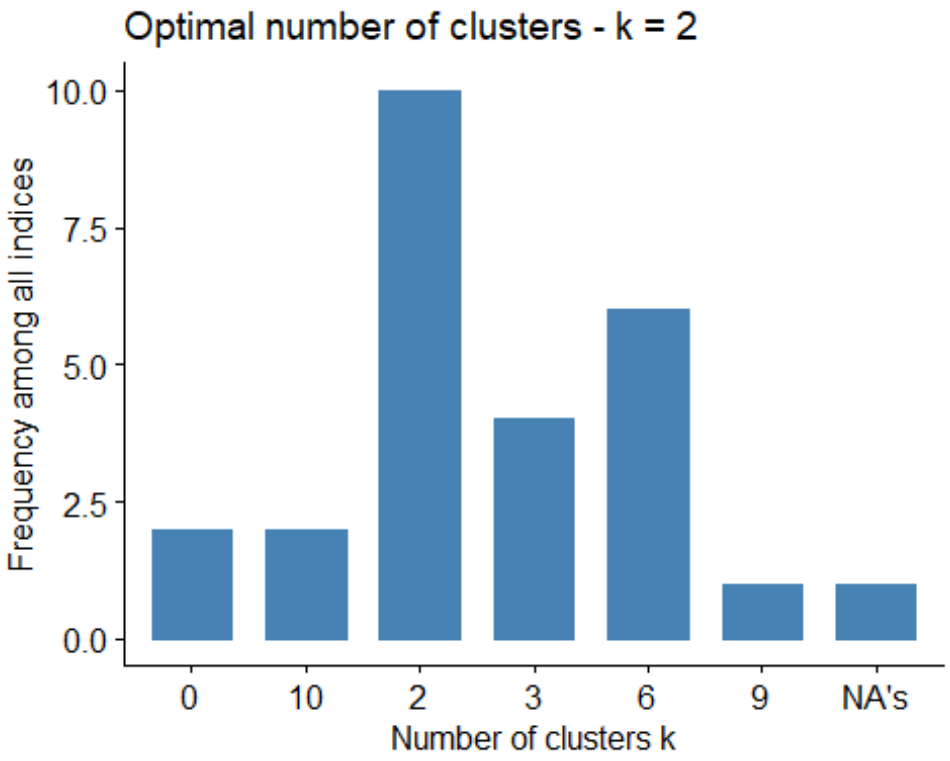

```
## *** : The Hubert index is a graphical method of determining the number
of clusters.
```

```
## In the plot of Hubert index, we seek a significant knee that correspon
ds to a significant increase of the value of the measure i.e the signific
ant peak in Hubert index second differences plot.
```



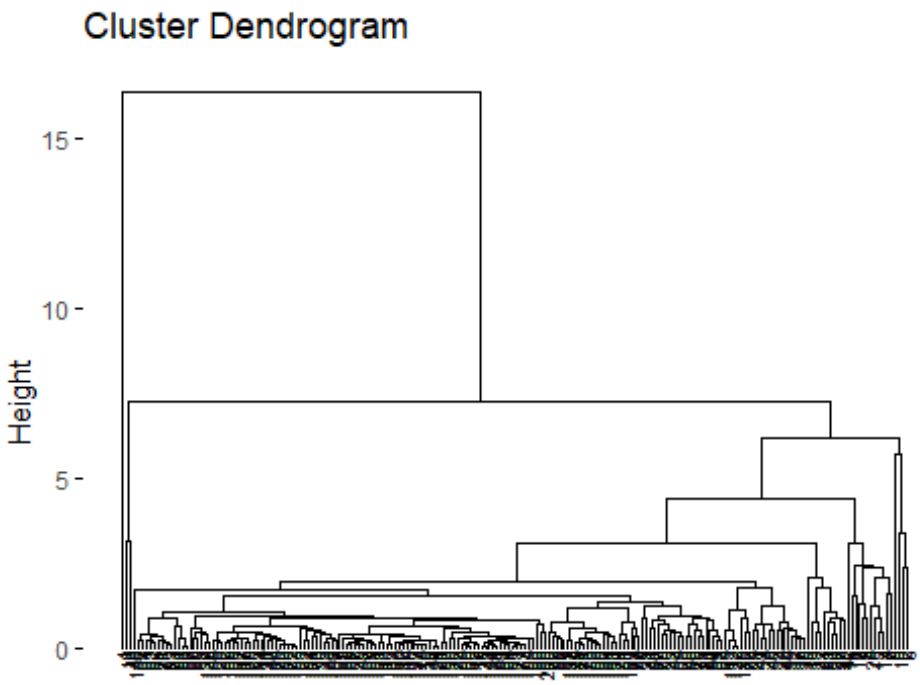

```
## *** : The D index is a graphical method of determining the number of c
lusters.
## In the plot of D index, we seek a significant knee (the significant pe
ak in Dindex second differences plot) that corresponds to a significant i
ncrease of the value of the measure.
##
## *******************************************************************
## * Among all indices:
## * 10 proposed 2 as the best number of clusters
## * 4 proposed 3 as the best number of clusters
## * 6 proposed 6 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************

fviz_nbclust(res.alm)
```

```
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 10 proposed  2 as the best number of clusters
## * 4 proposed  3 as the best number of clusters
## * 6 proposed  6 as the best number of clusters
## * 1 proposed  9 as the best number of clusters
## * 2 proposed  10 as the best number of clusters
## * 1 proposed  NA's as the best number of clusters
##
## Conclusion
## ========================
## * According to the majority rule, the best number of clusters is  2 .
```



According to the majority rule, the best number of clusters is 2 for this method.

```
res.hc2 <- hclust(dist.eucl.200, method = "average")
fviz_dend(res.hc2, cex = 0.5)
```



Looking at the dendogram the results is a little ambiguos, but we have some way to check if this is a good clustering solution, like check the correlation between the cophenetic distance and the original one.

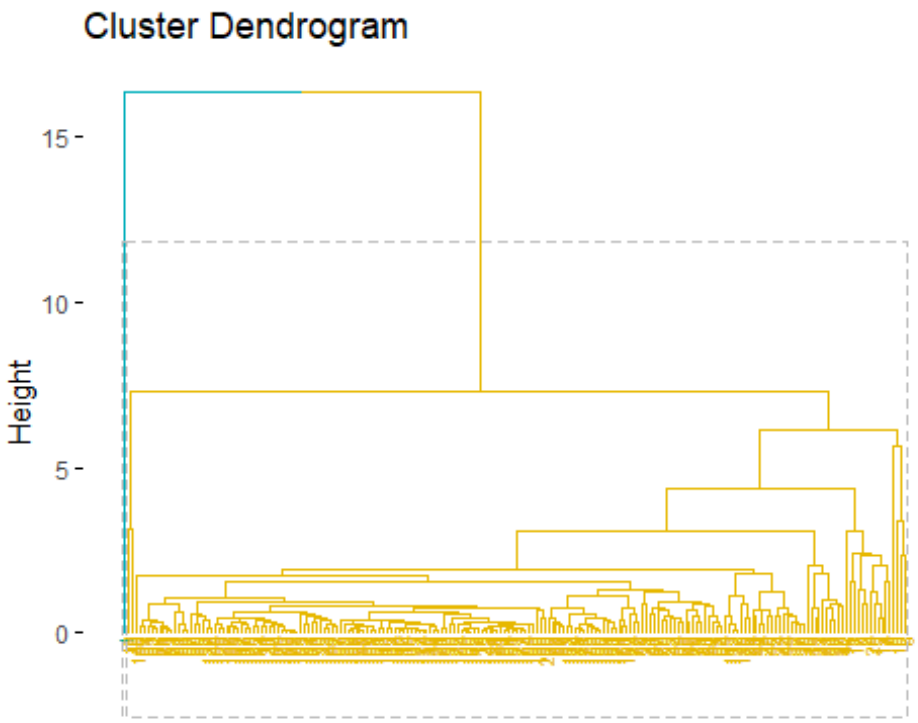
The closer the value of the correlation coefficient is to 1, the more accurately the clustering solution reflects our data. Values above 0.75 are felt to be good.

```
res.hc2 <- hclust(dist.eucl.200, method = "average")
cor(dist.eucl.200, cophenetic(res.hc2))

## [1] 0.9564332
```

The correlation coefficient shows that this clustering solution reflects our data and its a good one to use.

Now, if we want to see clusters, we have to cut the hierarchical tree, for example specifying the number of groups that we want.

```
d.alm <- cutree(res.hc2, k = 2)
fviz_dend(res.hc2, k = 2, cex = 0.5, k_colors = c("#00AFBB", "#E7B800"),
color_labels_by_k = TRUE, rect = TRUE)
```

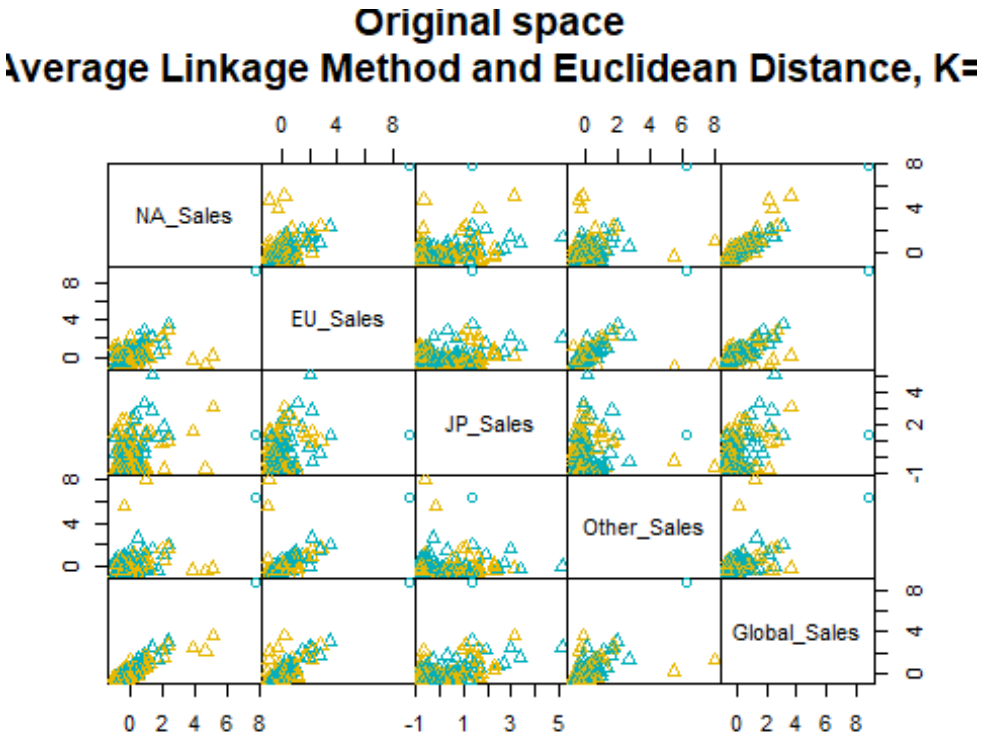



```
table(d.alm)

## d.alm
##   1   2
##   1 199
```

As we can see in the dendogram and in the table, the clusterization puts only 1 observation in the first cluster.

We can visualize these clustering results in the original space, via the matrix of pairwise scatterplots, using the *pairs()* function.

```
pairs(vg_sales.scaled, gap=0, pch=d.alm, col=c("#00AFBB", "#E7B800"),  ma
in="Original space\n- Average Linkage Method and Euclidean Distance, K=2"
[d.alm])
```

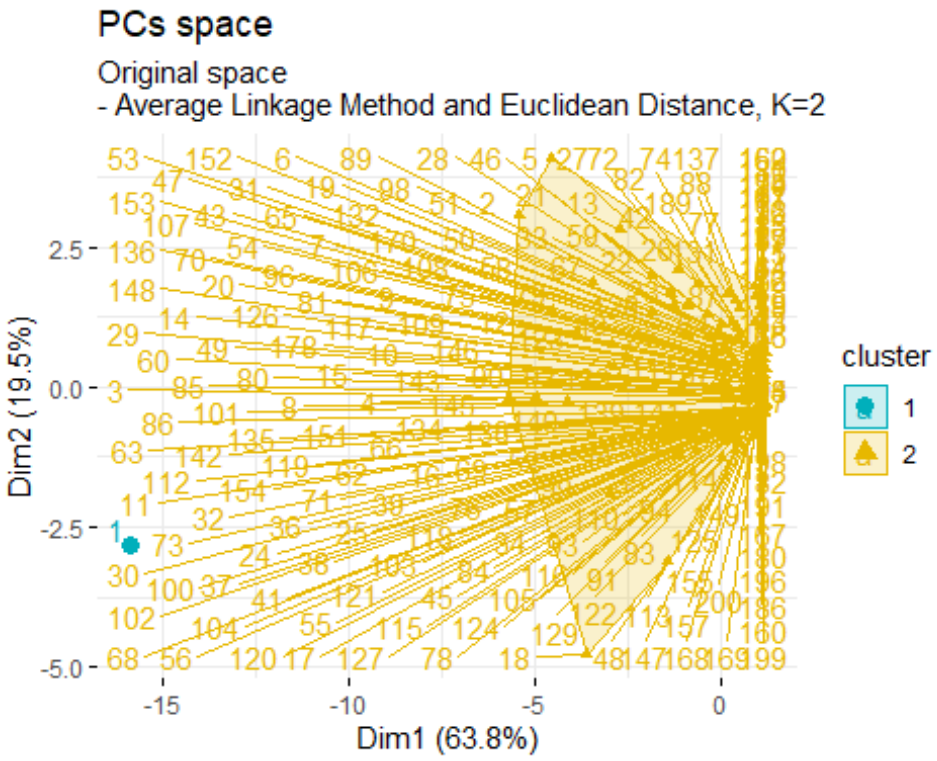


This is the original matrix of scatterplots, so we are using the original space, which is more important than the one spanned by the first 2 principal components, because we have searched the groups in the original space.

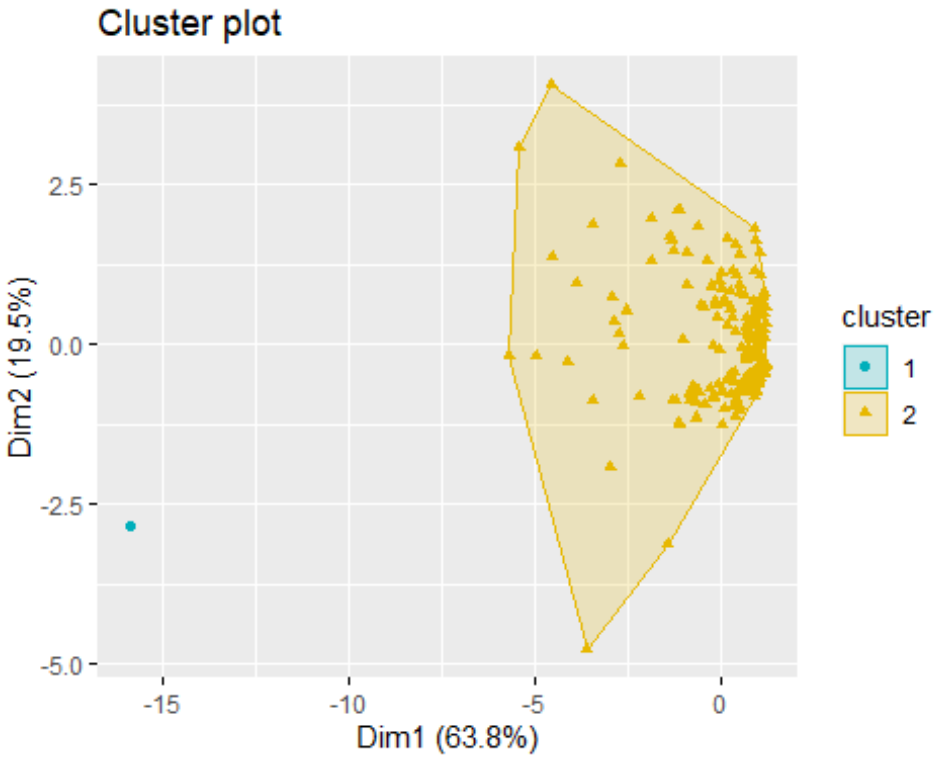
Looking to this graph we see that in the original space the observations overlap a lot.

Using the function fviz *cluster()*, we can also visualize the results in the scatter plot of the first 2 PCs.

```
fviz_cluster(list(data = vg_sales.scaled, cluster = d.alm), palette = c("
#00AFBB", "#E7B800"), ellipse.type = "convex", main="PCs space", repel =
TRUE,
show.clust.aver = FALSE, ggtheme = theme_minimal())+labs(
subtitle = "Original space\n- Average Linkage Method and Euclidean Distan
ce, K=2", cex.sub= 0.5)
```



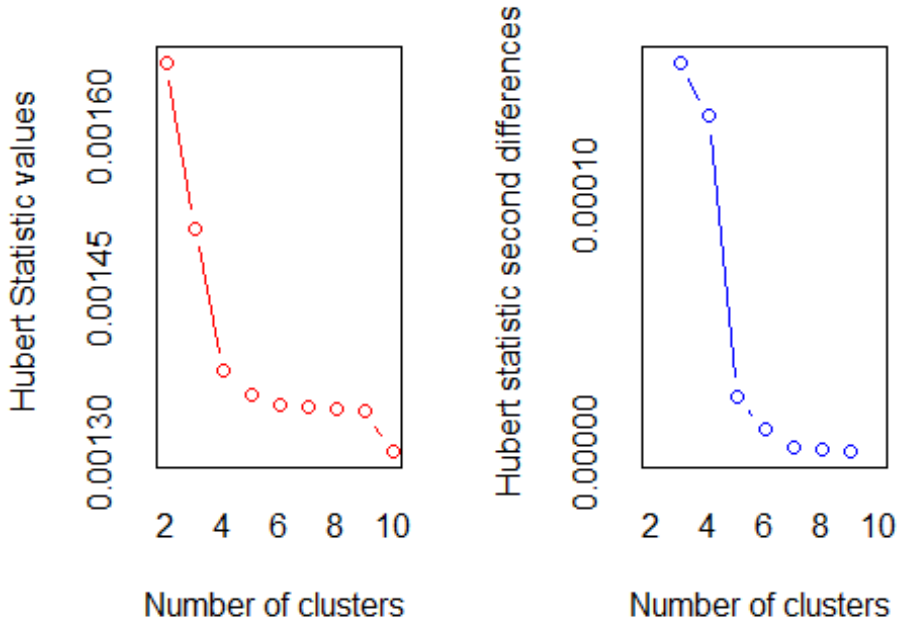

```
fviz_cluster(list(data=vg_sales.scaled, cluster =  d.alm),geom="point",el
lipse.type="convex",palette = c("#00AFBB", "#E7B800"),repel = TRUE, show.
clust.cent = FALSE)
```

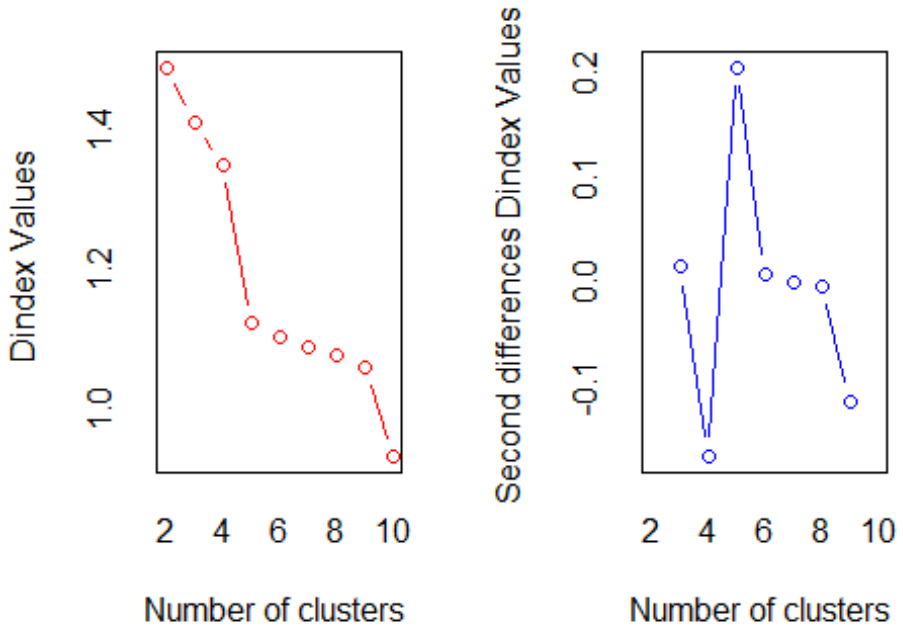


Average Linkage Method and Manhattan Distance

Now lets use the manhattan distance to do the Average Linkage Method:

```
library(NbClust)
res.alm.md <- NbClust(vg_sales.scaled, distance = "manhattan", min.nc = 2
, max.nc = 10,
method = "average")

## [1] "Frey index : No clustering structure in this data set"
```



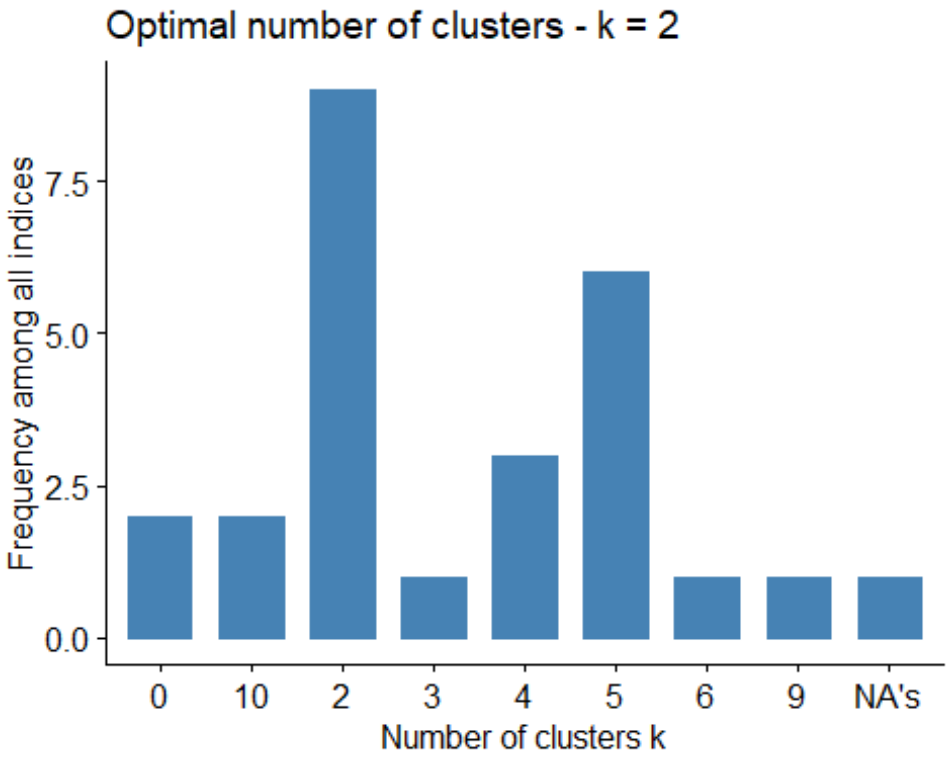



```
## *****************************************************************
## * Among all indices:
## * 9 proposed 2 as the best number of clusters
## * 1 proposed 3 as the best number of clusters
## * 3 proposed 4 as the best number of clusters
## * 6 proposed 5 as the best number of clusters
## * 1 proposed 6 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *****************************************************************
```

```
fviz_nbclust(res.alm.md)
```

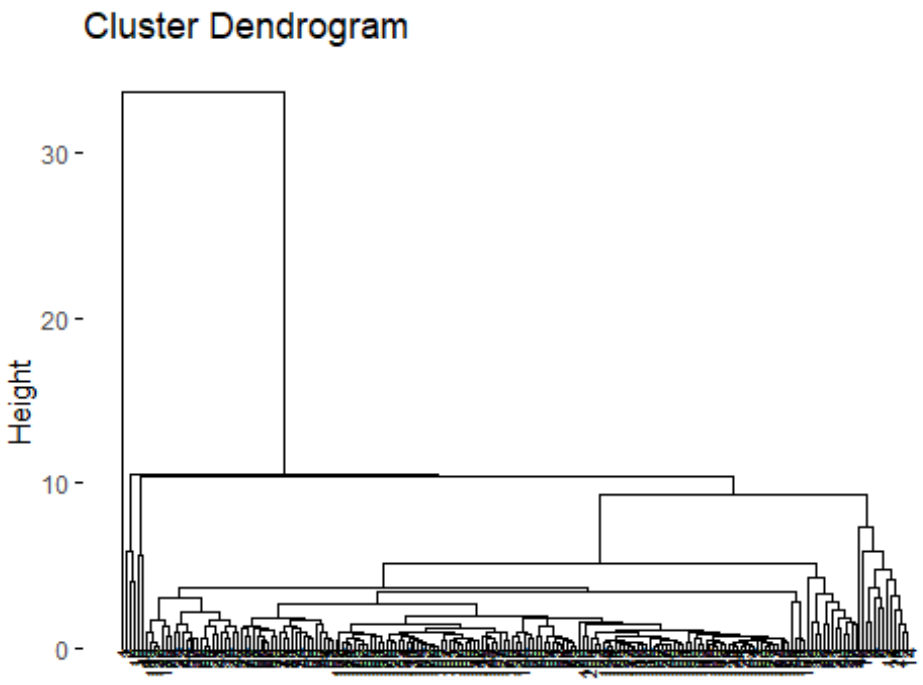
```
## Among all indices:
## ==================
## * 2 proposed  0 as the best number of clusters
## * 9 proposed  2 as the best number of clusters
## * 1 proposed  3 as the best number of clusters
## * 3 proposed  4 as the best number of clusters
## * 6 proposed  5 as the best number of clusters
```

```
## * 1 proposed  6 as the best number of clusters
## * 1 proposed  9 as the best number of clusters
## * 2 proposed  10 as the best number of clusters
## * 1 proposed  NA's as the best number of clusters
##
## Conclusion
## =========================
## * According to the majority rule, the best number of clusters is  2 .
```



According to the majority rule, the best number of clusters for this method is 2 .

```
res.alm.md1 <- hclust(dist.man.200, method = "average")
fviz_dend(res.alm.md1, cex = 0.5)
```



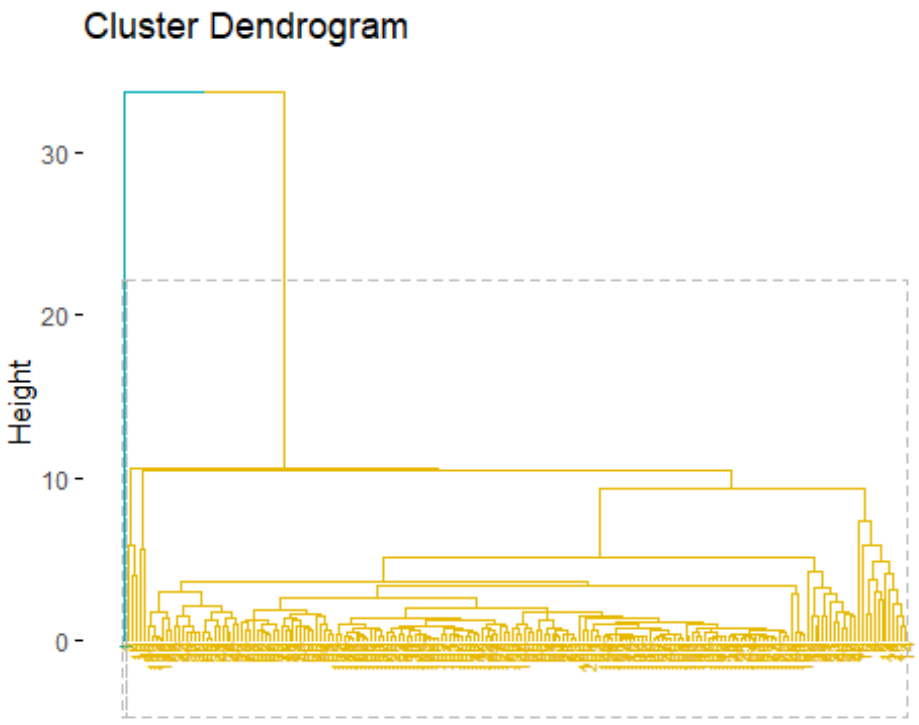
As we can see, one more time the result of the dendogram is ambiguos, so lets check if this is a good clustering solution using the cophenetic distance:

```
res.cd.alm.md <- hclust(dist.man.200, method = "average")
cor(dist.man.200, cophenetic(res.cd.alm.md ))

## [1] 0.9481732
```

The correlation coefficient of 0.94 shows that this clustering solution reflects our data and its a good one to use.

Now we need to cut the hierarchical tree:

```
alm.md <- cutree(res.cd.alm.md, k = 2)
fviz_dend(res.cd.alm.md, k = 2, cex = 0.5, k_colors = c("#00AFBB", "#E7B8
00"), color_labels_by_k = TRUE, rect = TRUE)
```

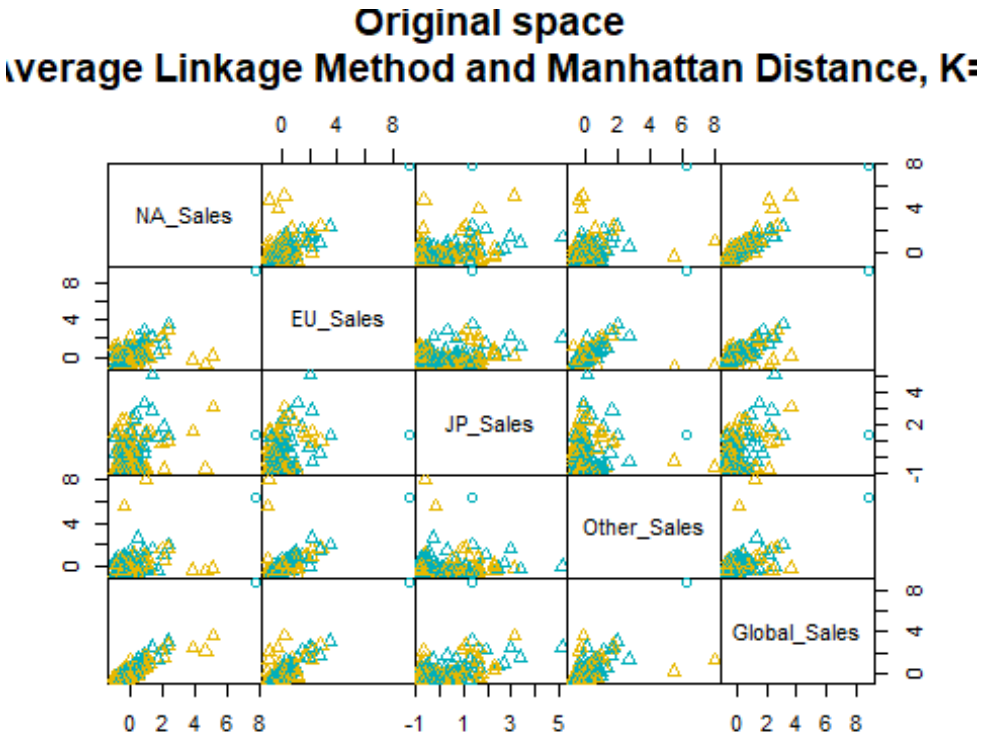



```
table(alm.md)

## alm.md
##   1   2
##   1 199
```

As we can see in the dendogram and in the table, the clusterization puts only 1 observation in the first once again, which means that the partioning is not good.

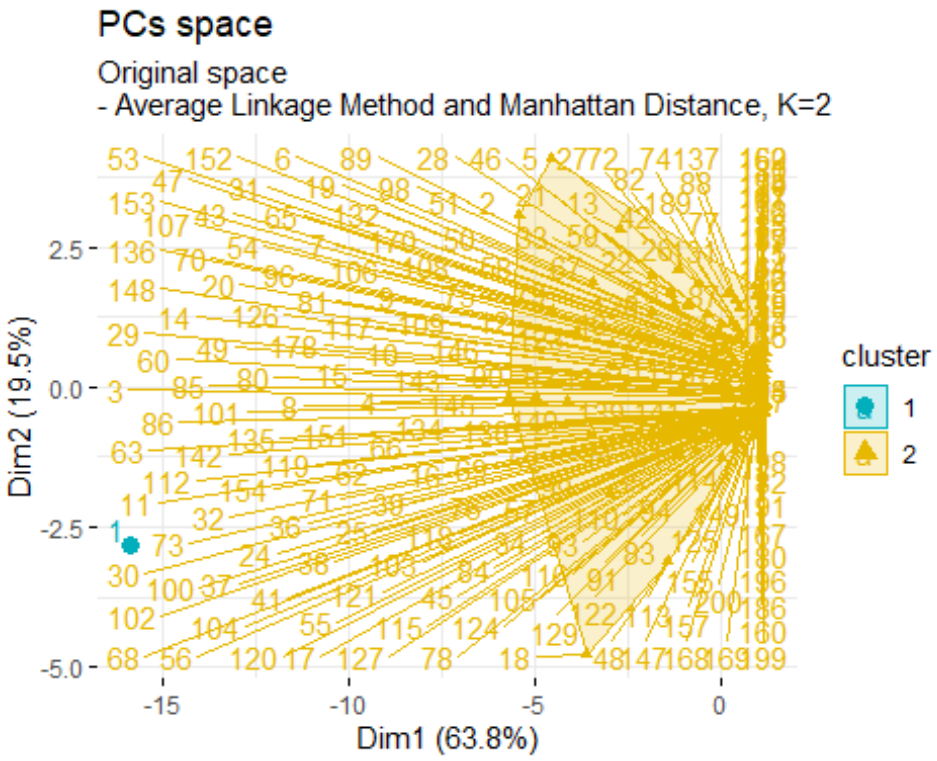
Now lets visualize the clustering results in the original space:

```
pairs(vg_sales.scaled, gap=0, pch=alm.md, col=c("#00AFBB", "#E7B800"),  m
ain="Original space\n- Average Linkage Method and Manhattan Distance, K=2
"[alm.md])
```

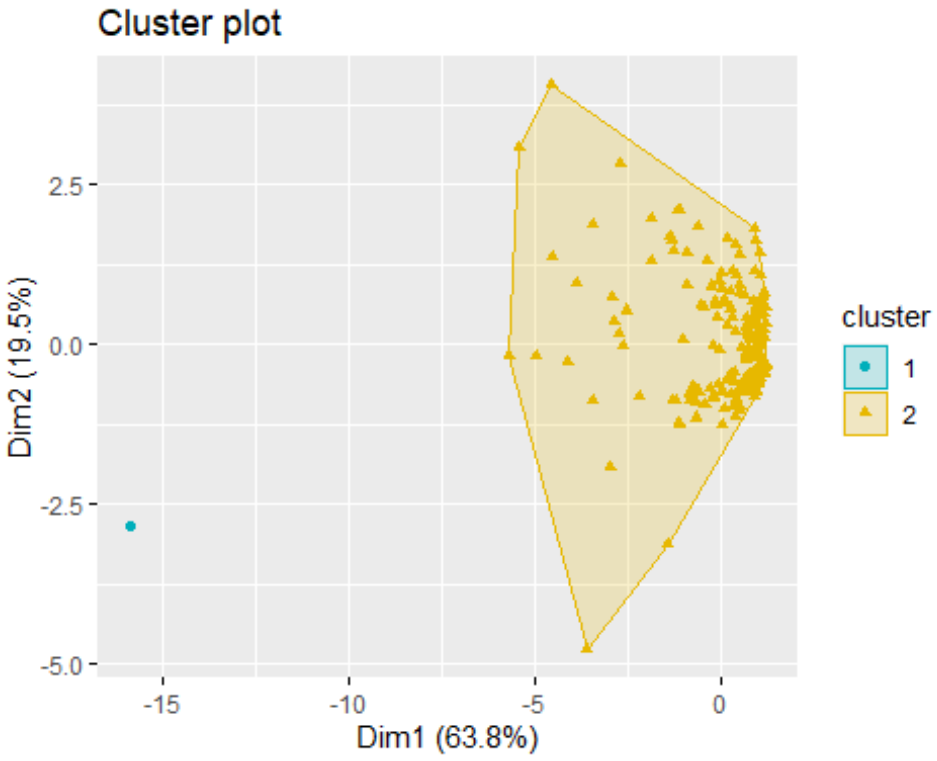


Now lets use the function fviz *cluster()*, to visualize the results in the scatter plot of the first 2 PCs.

```
fviz_cluster(list(data = vg_sales.scaled, cluster = alm.md), palette = c(
"#00AFBB", "#E7B800"), ellipse.type = "convex", main="PCs space", repel =
TRUE,
show.clust.aver = FALSE, ggtheme = theme_minimal())+labs(
subtitle = "Original space\n- Average Linkage Method and Manhattan Distan
ce, K=2", cex.sub= 0.5)
```




```
fviz_cluster(list(data=vg_sales.scaled, cluster =  alm.md),geom="point",ellipse.type="convex",palette = c("#00AFBB", "#E7B800"),repel = TRUE, show.clust.cent = FALSE)
```

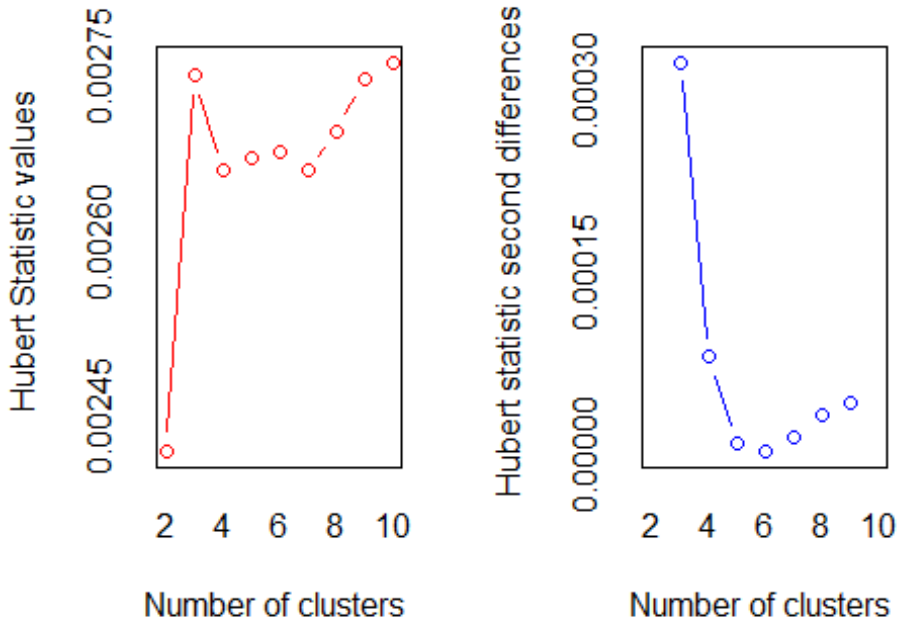


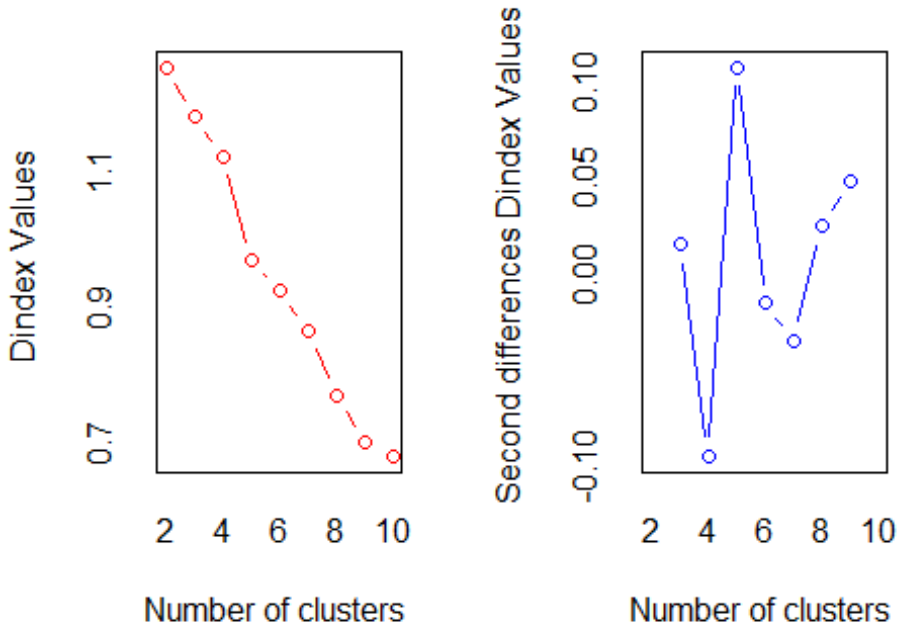
The results are pretty much alike with the Euclidean Distance, and even with the cophenetic results with a good coefficient, I don't believe that method give us a good partioning of the datas because we have the 1st cluster with only one observation.

Ward's Linkage Method and Euclidean Distance

This approach of calculating the similarity between the clusters is exactly the same as Average except that Ward's method calculates the sum of the square of the distances.

```
library(NbClust)
res.wlm1 <- NbClust(vg_sales.scaled, distance = "euclidean", min.nc = 2,
max.nc = 10,
method = "ward.D2")
```



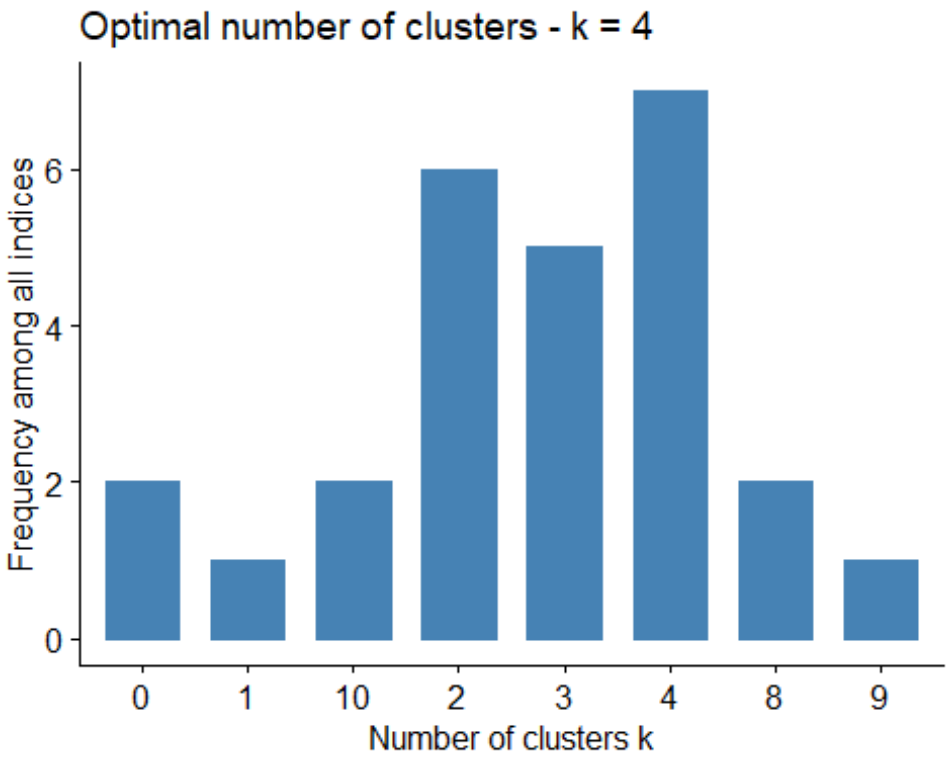



```
## *****************************************************************
## * Among all indices:
## * 6 proposed 2 as the best number of clusters
## * 5 proposed 3 as the best number of clusters
## * 7 proposed 4 as the best number of clusters
## * 2 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  4
##
##
## *****************************************************************
```

```
fviz_nbclust(res.wlm1)
```

```
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 1 proposed  1 as the best number of clusters
## * 6 proposed  2 as the best number of clusters
## * 5 proposed  3 as the best number of clusters
## * 7 proposed  4 as the best number of clusters
## * 2 proposed  8 as the best number of clusters
## * 1 proposed  9 as the best number of clusters
```

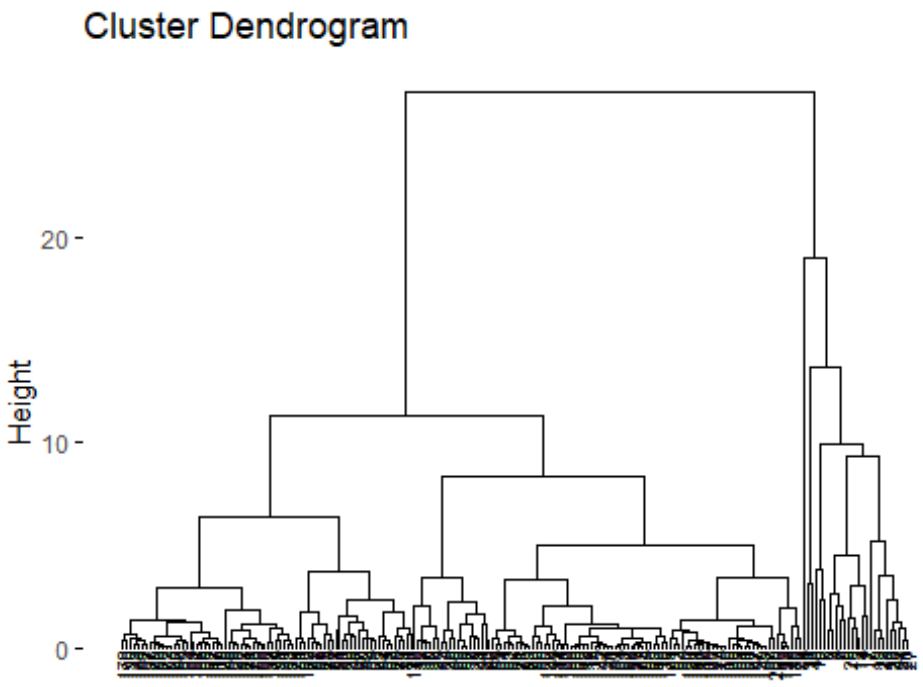
```
## * 2 proposed  10 as the best number of clusters
##
## Conclusion
## =========================
## * According to the majority rule, the best number of clusters is  4 .
```



According to the majority rule, the best number of clusters for this method 4 .

Let's draw a dendogram to visualize it better:

```
res.wlm <- hclust(d = dist.eucl.200, method = "ward.D2")
fviz_dend(res.wlm, cex = 0.5)
```



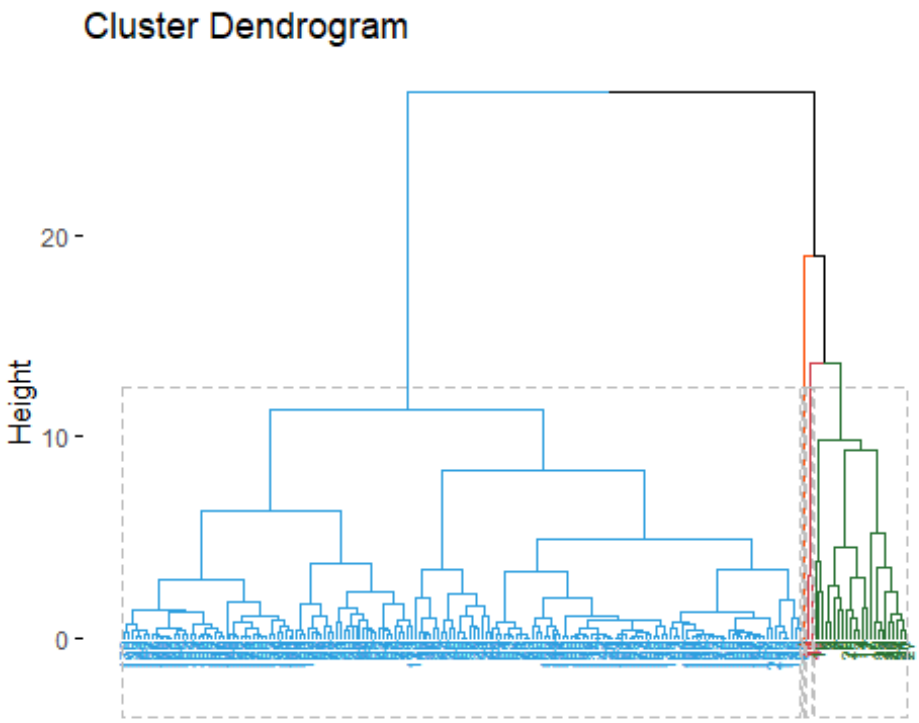
Now, let's check for the correlation between the cophenetic distance and the original one:

```
res.coph1 <- cophenetic(res.wlm)
cor(dist.eucl.200, res.coph1)

## [1] 0.6991678
```

The value of 0.69 is not low but its now good enough. This means that this clustering method does not preserve the true original distances between units, different from the Average Linkage Method.

Now to see clusters, we have to cut the hierarchical tree as well:

```
d.wlm <- cutree(res.wlm, k = 4)
fviz_dend(res.wlm, k = 4, cex = 0.5, k_colors = c("#2E9FDF", "#FC4E07", "
#CF3E4B", "#287233"), color_labels_by_k = TRUE, rect = TRUE)
```

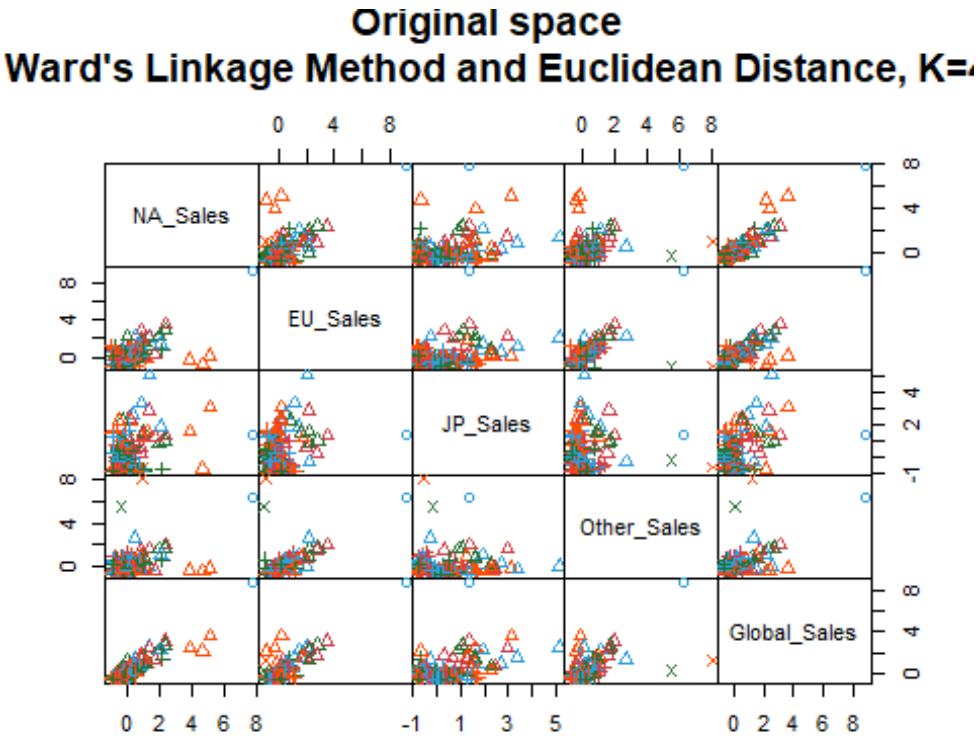



```
table(d.wlm)

## d.wlm
##   1   2   3   4
##   1  24 173   2
```

In this case we can see a more clear view and we have a better distribution of the clusters, the first cluster has 1 observation, the second one have 24 observations, the third has 173 observations and the fourth has 2.

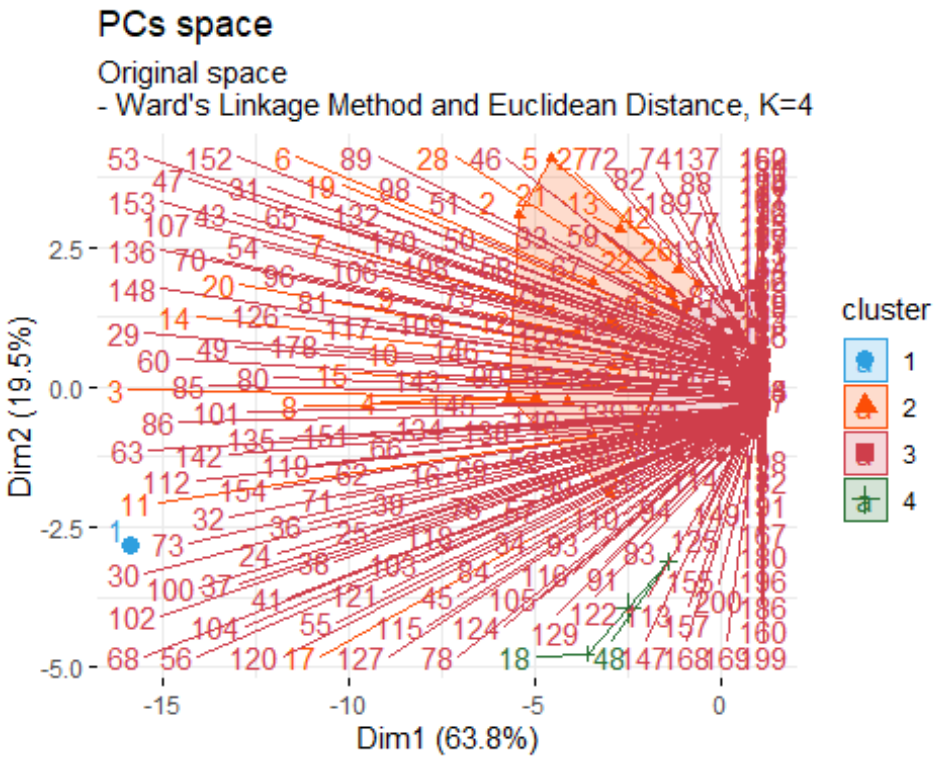
Now lets check it in the original space:

```
pairs(vg_sales.scaled, gap=0, pch=d.wlm, col=c("#2E9FDF", "#FC4E07", "#CF
3E4B", "#287233"),  main="Original space\n- Ward's Linkage Method and Euc
lidean Distance, K=4"[d.wlm])
```

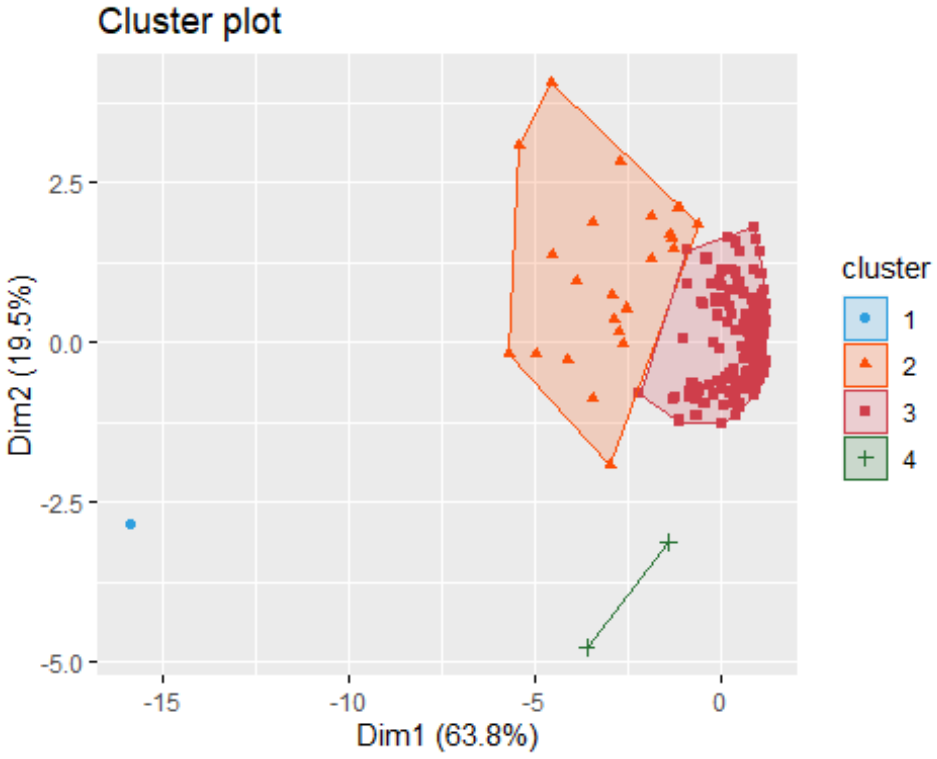


Now lets visualize the scatter plott of the 4 PCs.

```
fviz_cluster(list(data = vg_sales.scaled, cluster = d.wlm), palette = c("
#2E9FDF", "#FC4E07", "#CF3E4B", "#287233"), ellipse.type = "convex", main
="PCs space", repel = TRUE,
show.clust.aver = FALSE, ggtheme = theme_minimal())+labs(
subtitle = "Original space\n- Ward's Linkage Method and Euclidean Distanc
e, K=4", cex.sub= 0.5)
```




```
fviz_cluster(list(data=vg_sales.scaled, cluster =  d.wlm),geom="point",el
lipse.type="convex",palette = c("#2E9FDF", "#FC4E07", "#CF3E4B", "#287233
"),repel = TRUE, show.clust.cent = FALSE)
```

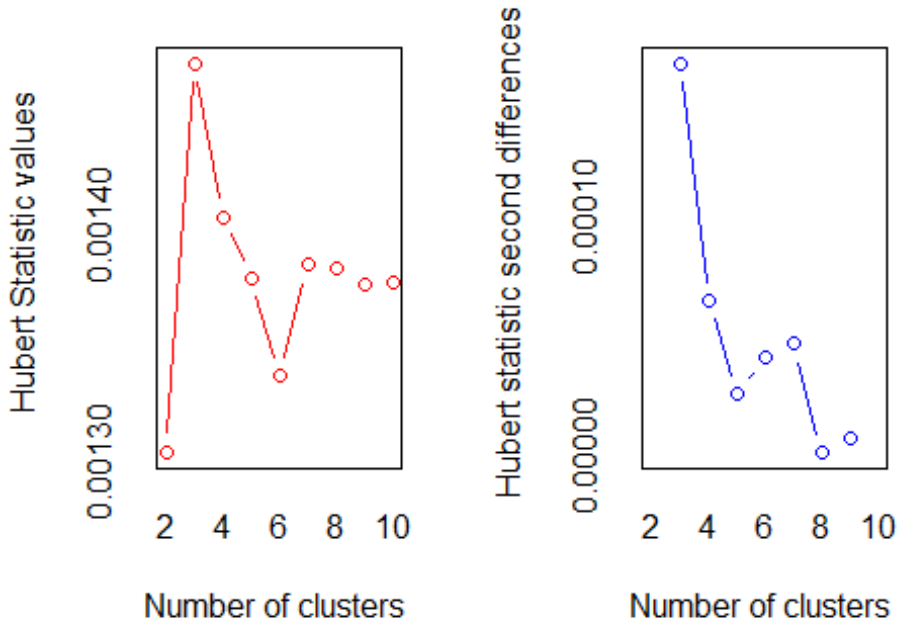


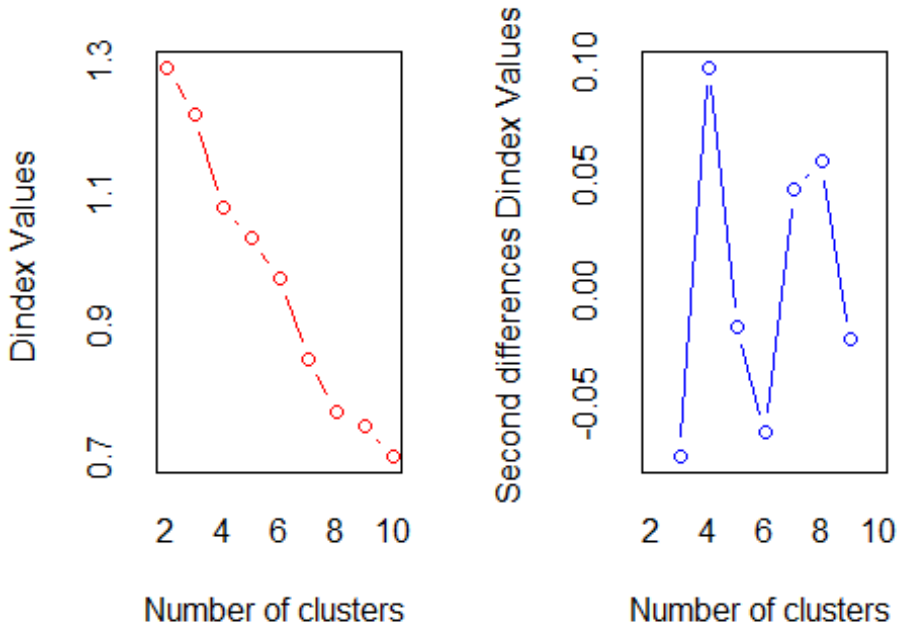
Even with the cophenetic coefficient not good enough, the Ward's Method gave us a better partioning of the clusters (even if with a lot observations in the third cluster) that the Average Linkage Method.

Ward's Linkage Method and the Manhattan Distance

Now lets try the Ward's method using the Manhattan distance:

```
library(NbClust)
res.wlm.md <- NbClust(vg_sales.scaled, distance = "manhattan", min.nc = 2
, max.nc = 10,
method = "ward.D2")
```



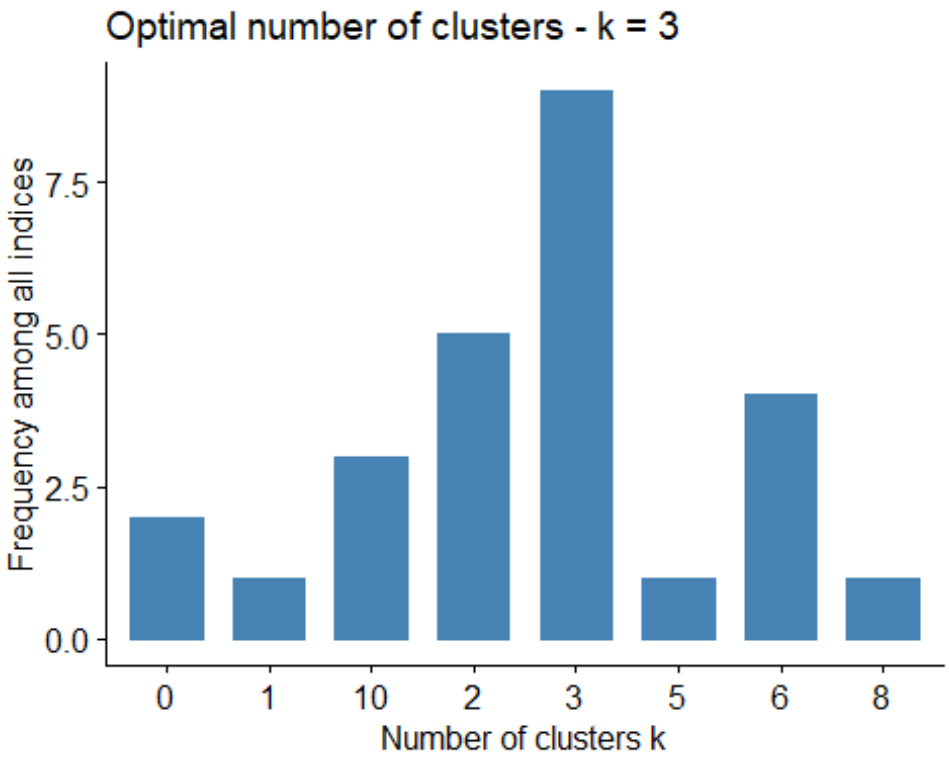



```
## *****************************************************************
## * Among all indices:
## * 5 proposed 2 as the best number of clusters
## * 9 proposed 3 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 4 proposed 6 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 3 proposed 10 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  3
##
##
## *****************************************************************
```

```
fviz_nbclust(res.wlm.md)
```

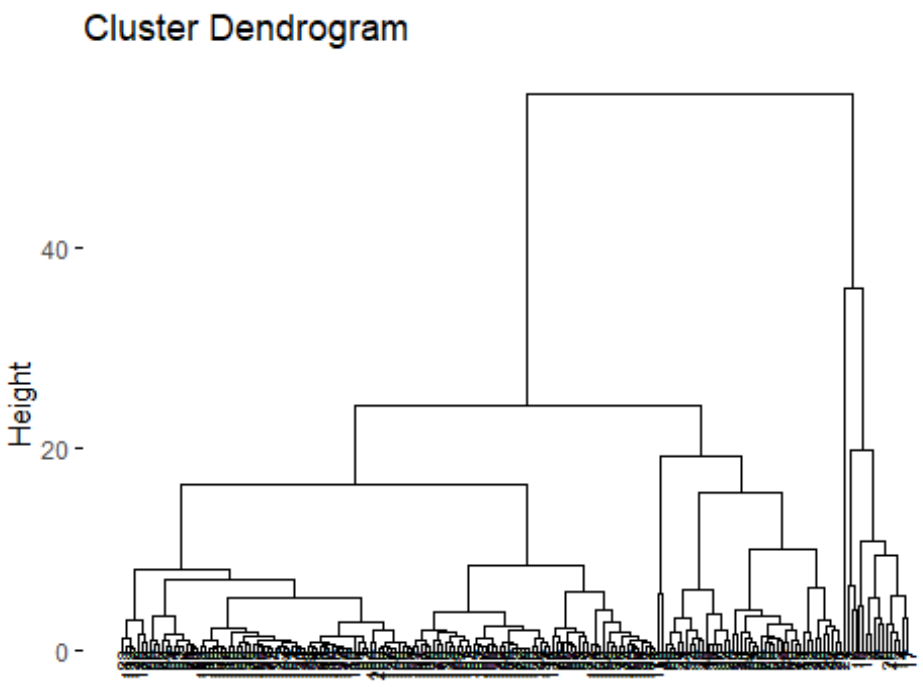
```
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 1 proposed  1 as the best number of clusters
## * 5 proposed  2 as the best number of clusters
## * 9 proposed  3 as the best number of clusters
## * 1 proposed  5 as the best number of clusters
## * 4 proposed  6 as the best number of clusters
```

```
## * 1 proposed  8 as the best number of clusters
## * 3 proposed  10 as the best number of clusters
##
## Conclusion
## =========================
## * According to the majority rule, the best number of clusters is  3 .
```



According to the majority rule, the best number of clusters is 3 .

```
res.wlm.md1 <- hclust(d = dist.man.200, method = "ward.D2")
fviz_dend(res.wlm.md1, cex = 0.5)
```



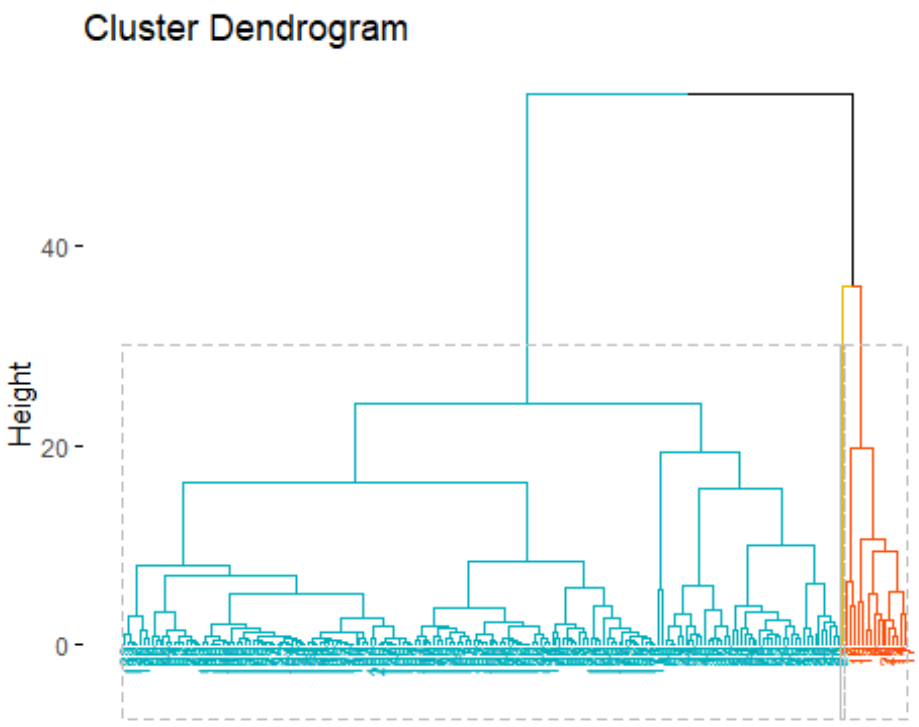
Now, let's check for the correlation between the cophenetic distance and the original one:

```
res.coph2 <- cophenetic(res.wlm.md1)
cor(dist.man.200, res.coph2)

## [1] 0.7221997
```

The value of 0.72 its close to the minimun of 0.75 but still isnt good enough. This means that this clustering method does not preserve the true original distances between units and the average linkage method is still with the best value in this case. But, onto the Ward's Method, using the Manhattan Distance seems to be more acurately.

Now lets cut the hierarchical tree:

```
d.wlm.md1 <- cutree(res.wlm.md1, k = 3)
fviz_dend(res.wlm.md1, k = 3, cex = 0.5, k_colors = c("#00AFBB", "#E7B800
", "#FC4E07"), color_labels_by_k = TRUE, rect = TRUE)
```

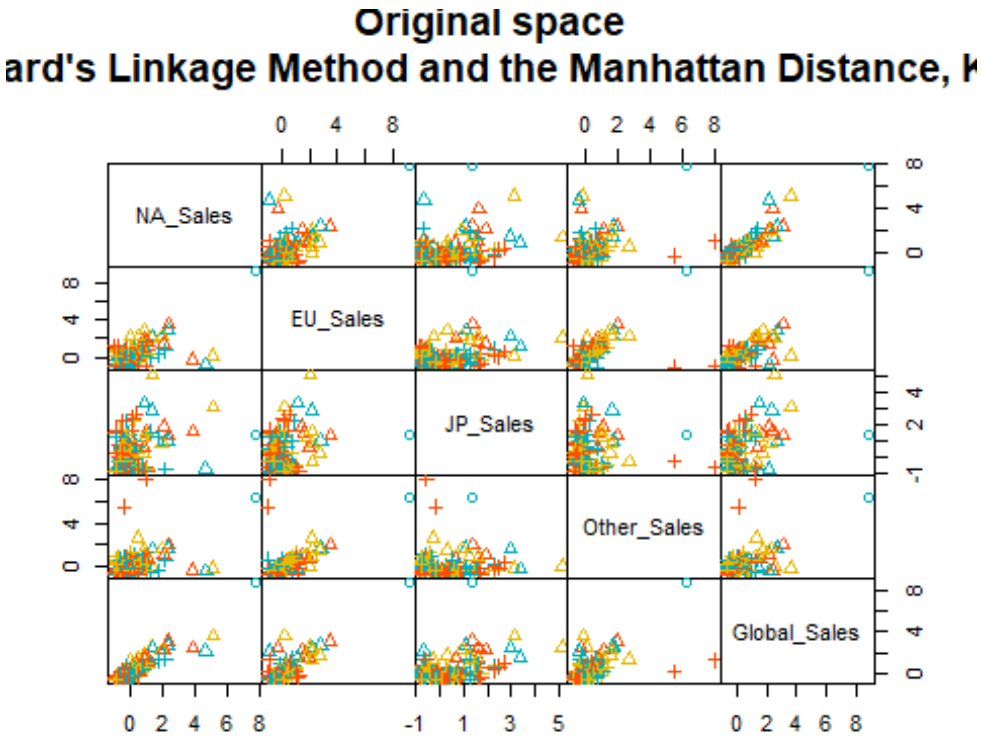



```
table(d.wlm.md1)

## d.wlm.md1
##   1   2   3
##   1  16 183
```

As we can see from the image and from table, this method suggest 3 clusters, but the first one contains only one observation, the second one 16 observations and the third contains 183 observations.

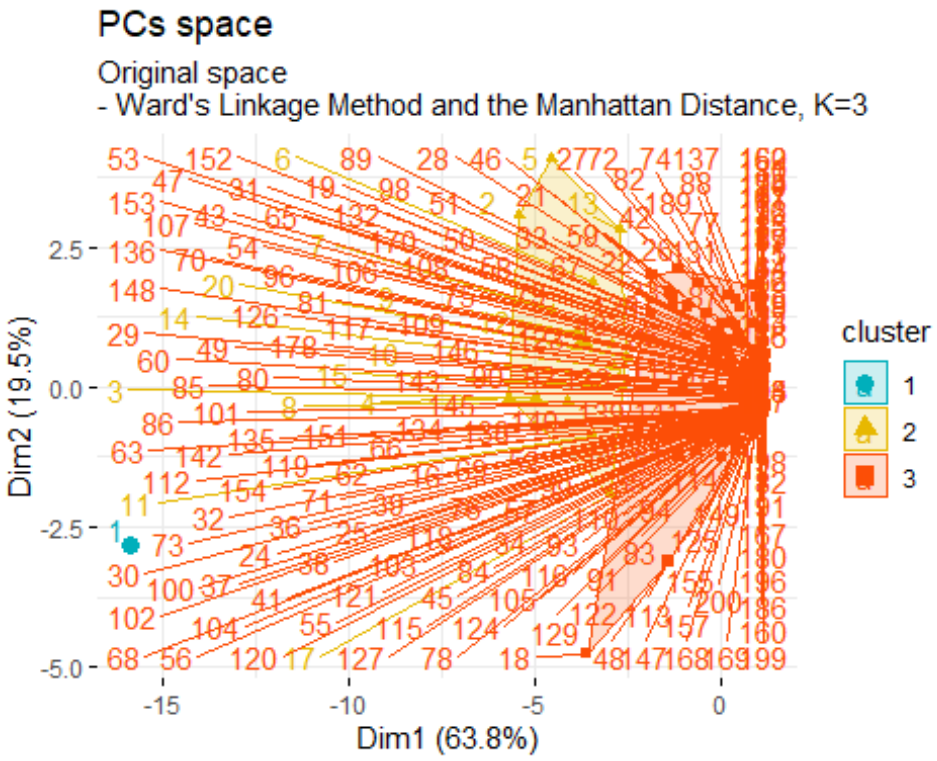
Now lets visualize the results in the original space:

```
pairs(vg_sales.scaled, gap=0, pch=d.wlm.md1, col=c("#00AFBB", "#E7B800",
"#FC4E07"),  main="Original space\n- Ward's Linkage Method and the Manhat
tan Distance, K=3"[d.wlm.md1])
```

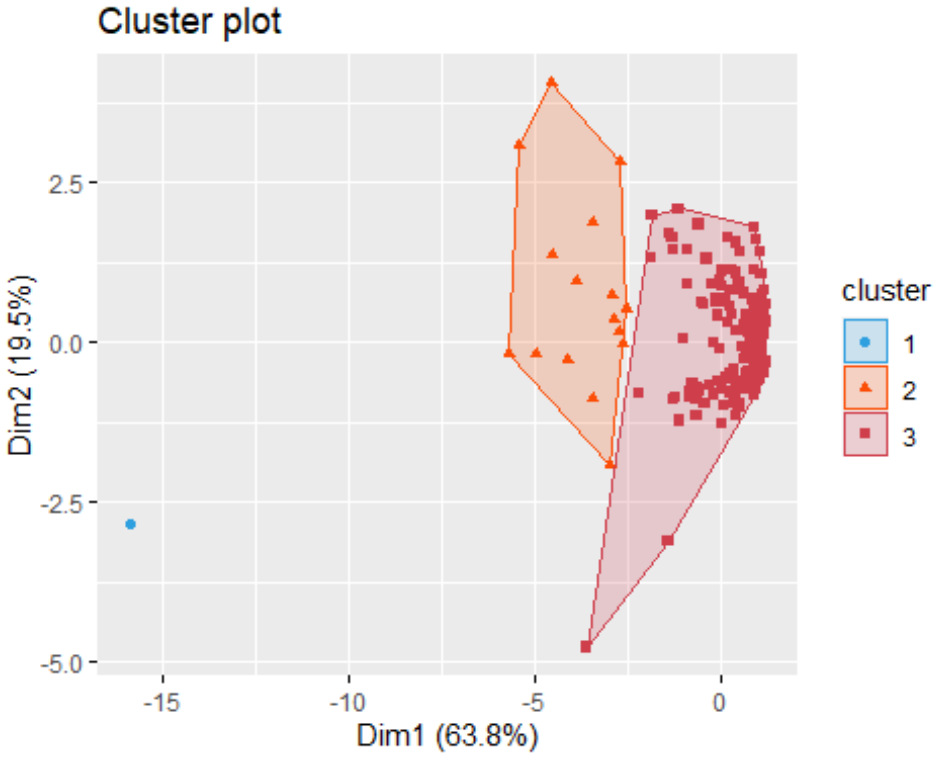


Now lets visualize the scatter plott of the 3 PCs.

```
fviz_cluster(list(data = vg_sales.scaled, cluster = d.wlm.md1), palette =
c("#00AFBB", "#E7B800", "#FC4E07"), ellipse.type = "convex", main="PCs sp
ace", repel = TRUE,
show.clust.aver = FALSE, ggtheme = theme_minimal())+labs(
subtitle = "Original space\n- Ward's Linkage Method and the Manhattan Dis
tance, K=3", cex.sub= 0.5)
```




```
fviz_cluster(list(data=vg_sales.scaled, cluster =  d.wlm.md1),geom="point
",ellipse.type="convex",palette = c("#2E9FDF", "#FC4E07", "#CF3E4B"),repe
l = TRUE, show.clust.cent = FALSE)
```

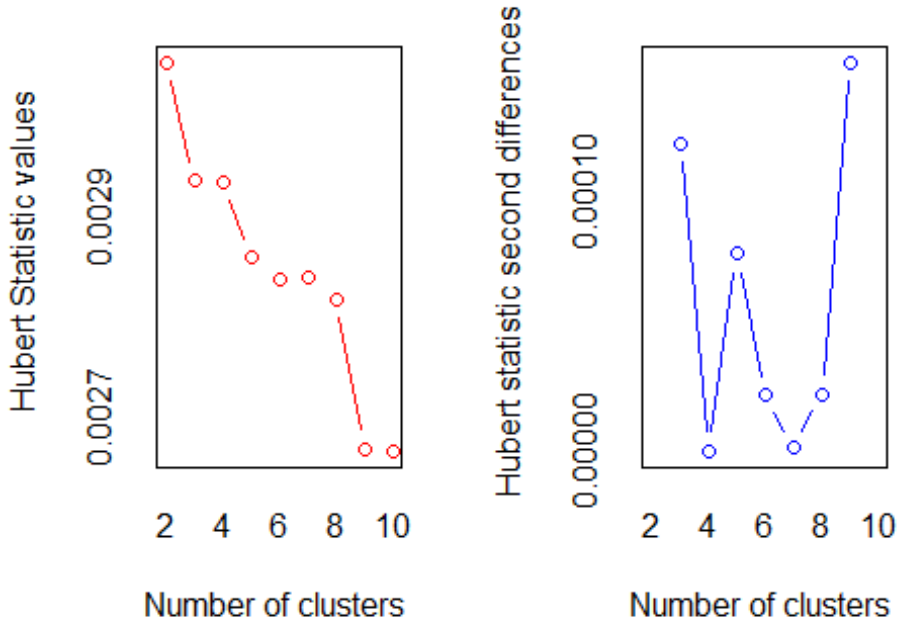


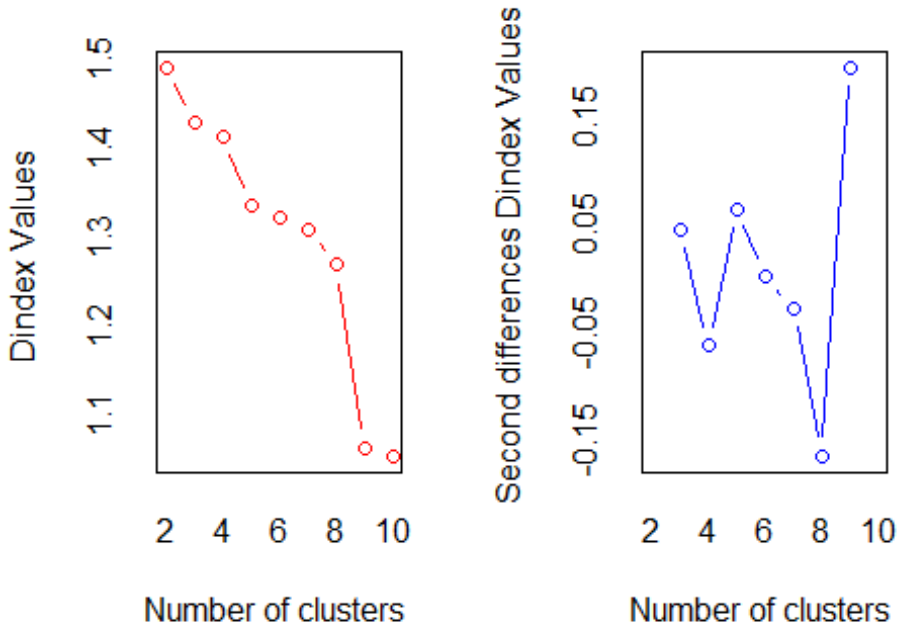
The Ward's method using the Manhattan distance have a better cophenetic coefficient, but still doesnt bring us a good partioning of the clusters because we still have clusters with only one observation.

Single Linkage Method and Euclidean Distance

Now we will use the single linkage method that it is a similarity measure between two clusters defined by shortest distance from any point in the 1st cluster to any point in the 2nd cluster.

```
library(NbClust)
res.slm.ed <- NbClust(vg_sales.scaled, distance = "euclidean", min.nc = 2
, max.nc = 10,
method = "single")
```



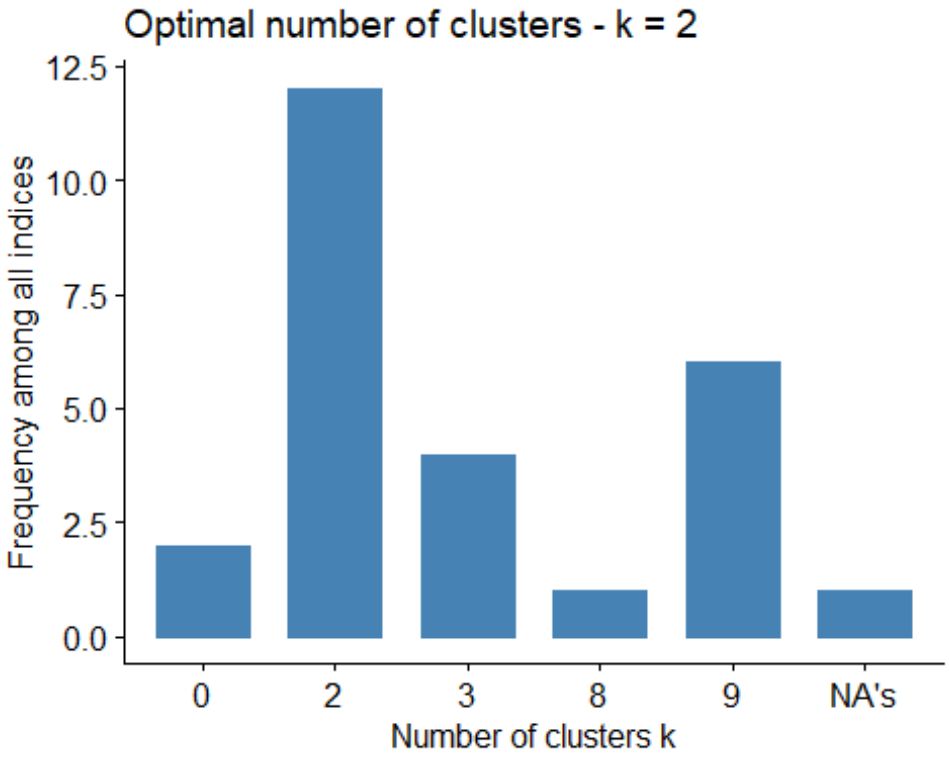



```
## *****************************************************************
## * Among all indices:
## * 12 proposed 2 as the best number of clusters
## * 4 proposed 3 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 6 proposed 9 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *****************************************************************
```

```
fviz_nbclust(res.slm.ed)
```

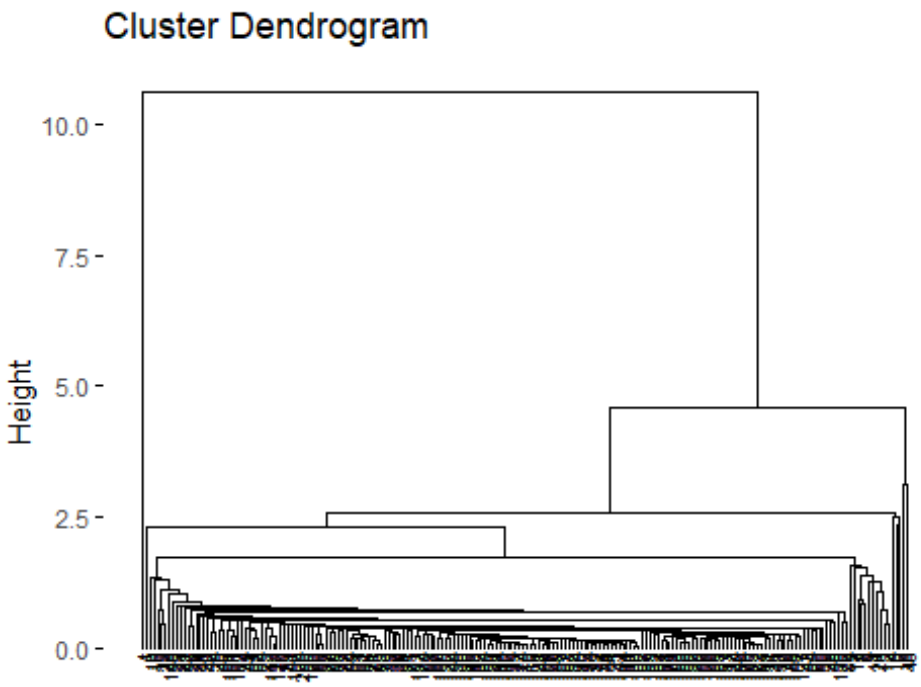
```
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 12 proposed  2 as the best number of clusters
## * 4 proposed  3 as the best number of clusters
## * 1 proposed  8 as the best number of clusters
## * 6 proposed  9 as the best number of clusters
## * 1 proposed  NA's as the best number of clusters
##
## Conclusion
```

```
## ************************
## * According to the majority rule, the best number of clusters is  2 .
```



According to the majority rule, the best number of clusters is 2.

```
res.slm.ed1 <- hclust(d = dist.eucl.200, method = "single")
fviz_dend(res.slm.ed1, cex = 0.5)
```



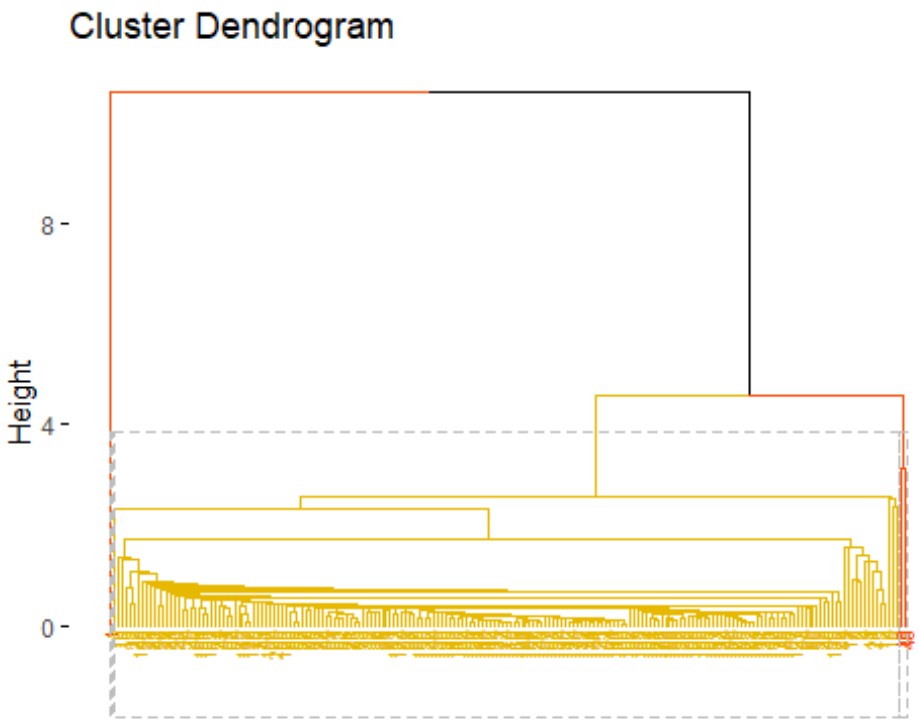
Now, let's check for the correlation between the cophenetic distance and the original one:

```
res.coph.sm.ed <- cophenetic(res.slm.ed1)
cor(dist.eucl.200, res.coph.sm.ed)

## [1] 0.919205
```

The value of 0.91 is a really good one this coefficient shows that the clustering solution reflects our data in a acurately way.

Now lets cut the hierarchical tree:

```
res.slm.ed2 <- cutree(res.slm.ed1, k = 2)
fviz_dend(res.slm.ed1, k = 3, cex = 0.5, k_colors = c("#FC4E07", "#E7B800
"), color_labels_by_k = TRUE, rect = TRUE)
```

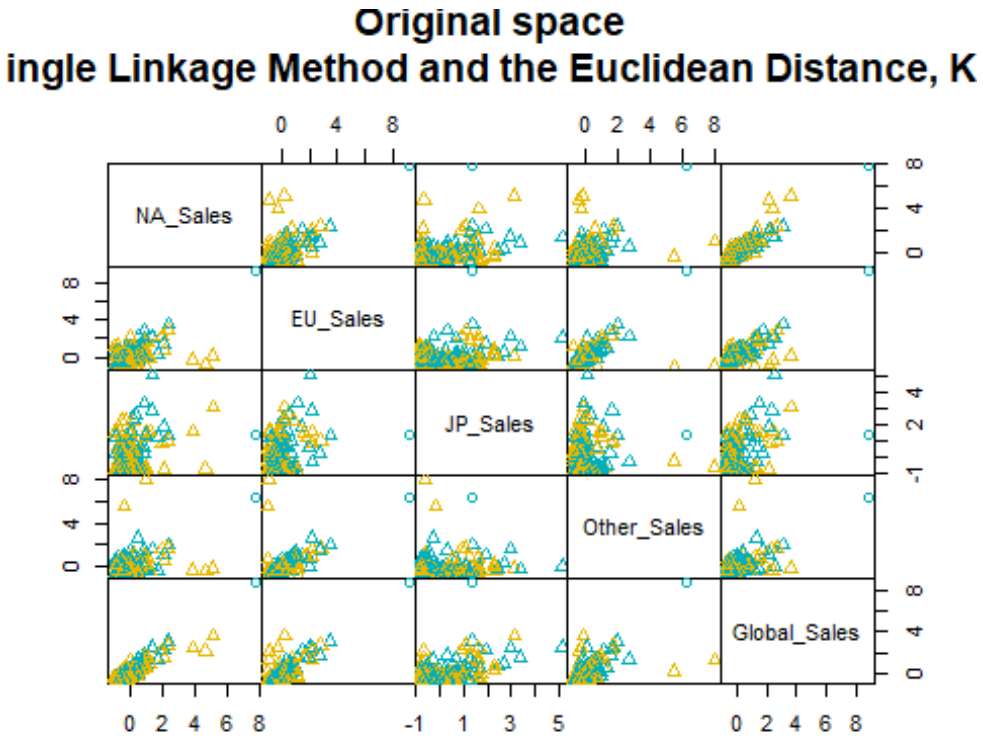



```
table(res.slm.ed2)

## res.slm.ed2
##   1   2
##   1 199
```

As we can see, the data partitioning is not good in this case and almost all the observations are concentred in one single cluster.

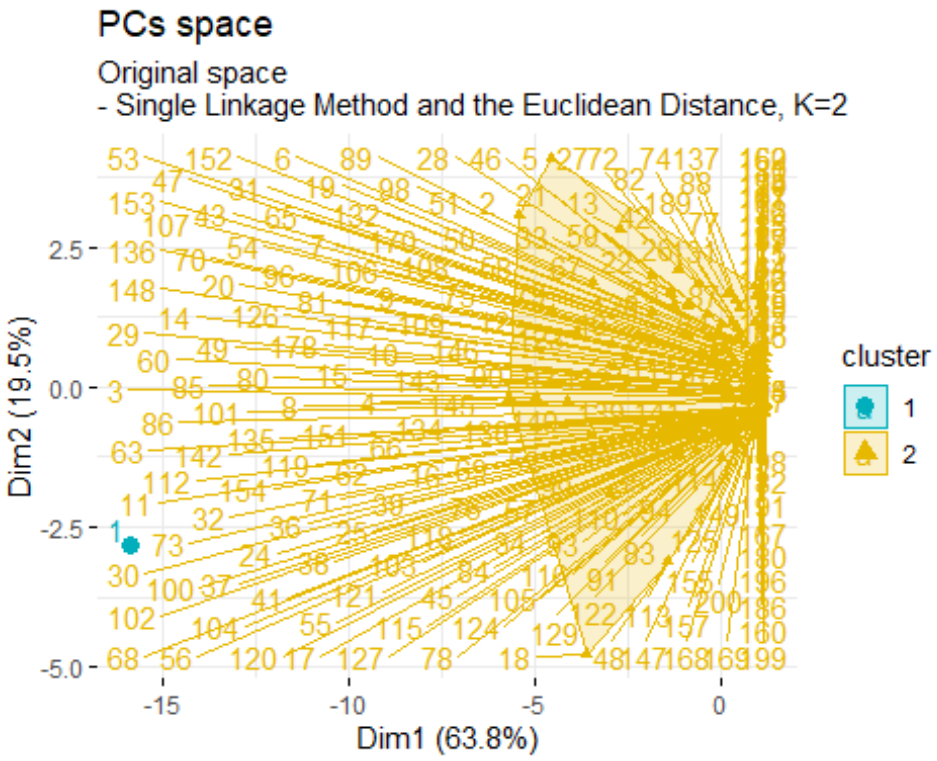
Now lets visualize the results in the original space:

```
pairs(vg_sales.scaled, gap=0, pch=res.slm.ed2, col=c("#00AFBB", "#E7B800"
),  main="Original space\n- Single Linkage Method and the Euclidean Dista
nce, K=2"[res.slm.ed2])
```

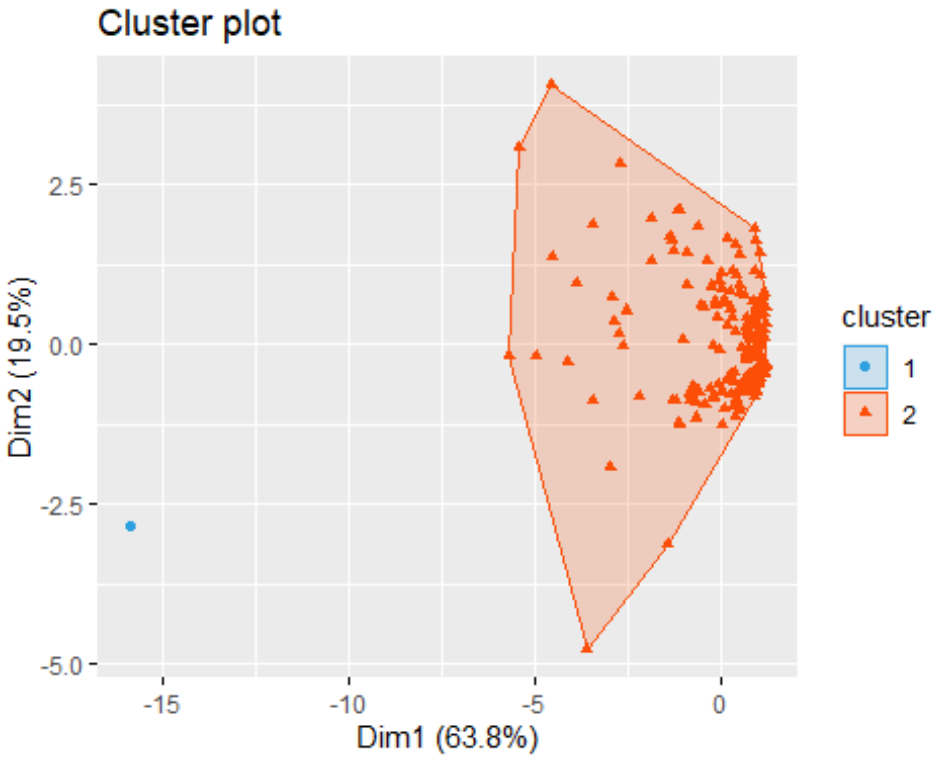


Now lets visualize the scatter plott of the 2 PCs.

```
fviz_cluster(list(data = vg_sales.scaled, cluster = res.slm.ed2), palette
= c("#00AFBB", "#E7B800"), ellipse.type = "convex", main="PCs space", rep
el = TRUE,
show.clust.aver = FALSE, ggtheme = theme_minimal())+labs(
subtitle = "Original space\n- Single Linkage Method and the Euclidean Dis
tance, K=2", cex.sub= 0.5)
```




```
fviz_cluster(list(data=vg_sales.scaled, cluster = res.slm.ed2),geom="poin
t",ellipse.type="convex",palette = c("#2E9FDF", "#FC4E07"),repel = TRUE,
show.clust.cent = FALSE)
```

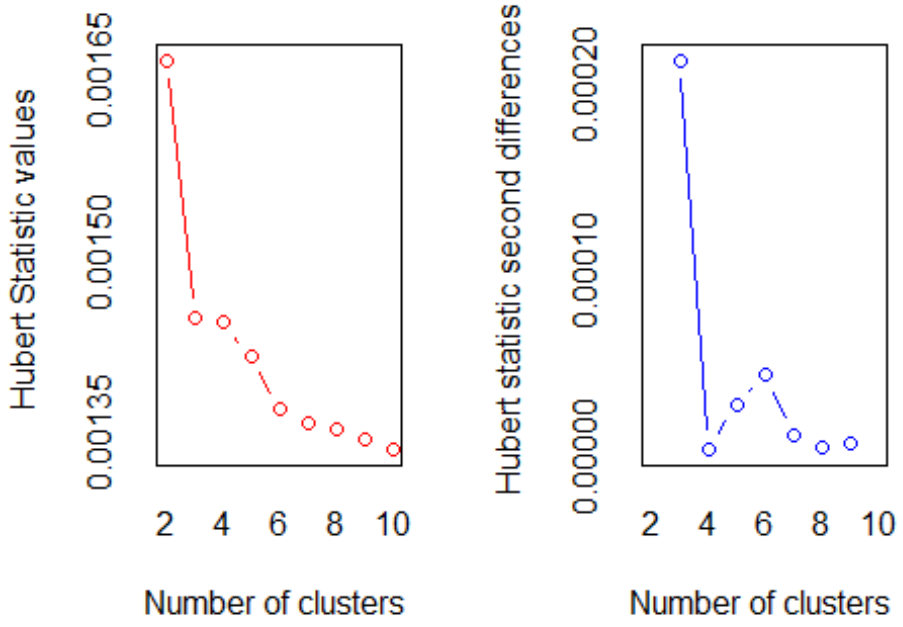


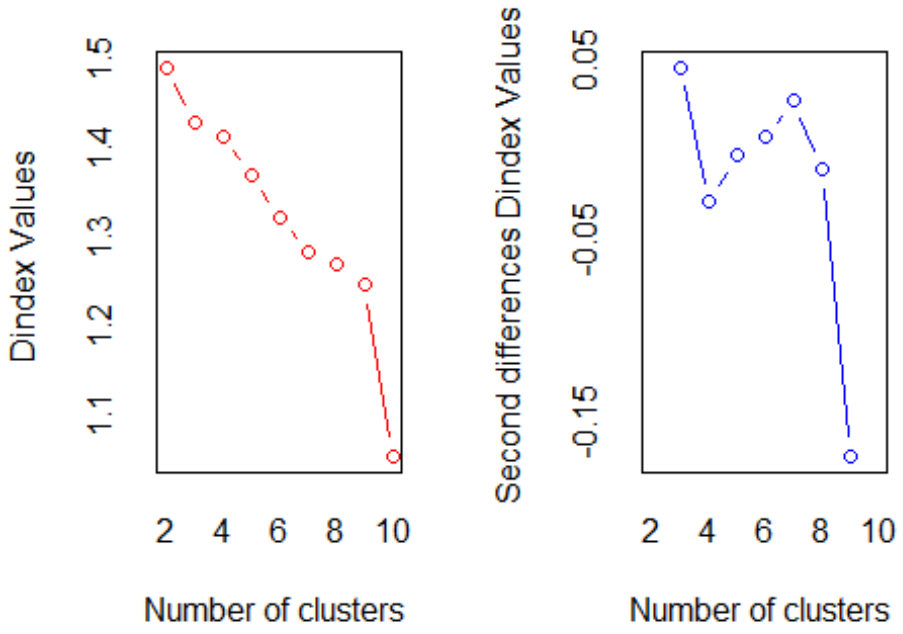
As we can see from this result, the single linkage method using euclidean distance and the average linked method suggest exactly the same partitioning in the observations.

Single Linkage Method and Manhattan Distance

Now we will do the single linkage method using the manhattan distance:

```
library(NbClust)
slm.md <- NbClust(vg_sales.scaled, distance = "manhattan", min.nc = 2, ma
x.nc = 10,
method = "single")

## [1] "Frey index : No clustering structure in this data set"
```



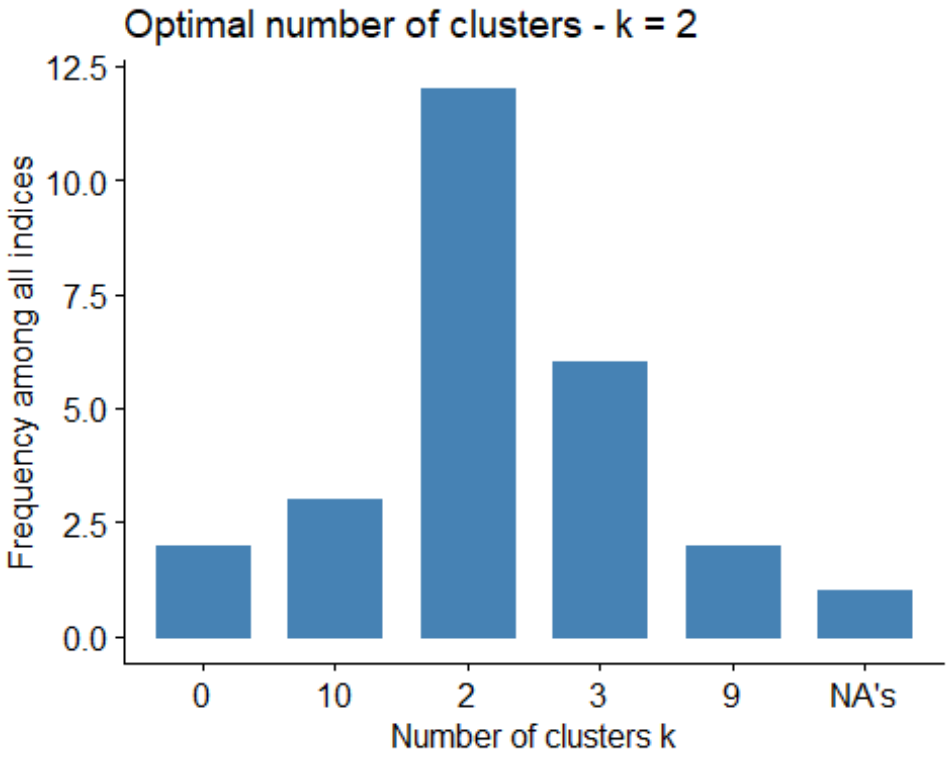



```
## *****************************************************************
## * Among all indices:
## * 12 proposed 2 as the best number of clusters
## * 6 proposed 3 as the best number of clusters
## * 2 proposed 9 as the best number of clusters
## * 3 proposed 10 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *****************************************************************
```

```
fviz_nbclust(slm.md)
```

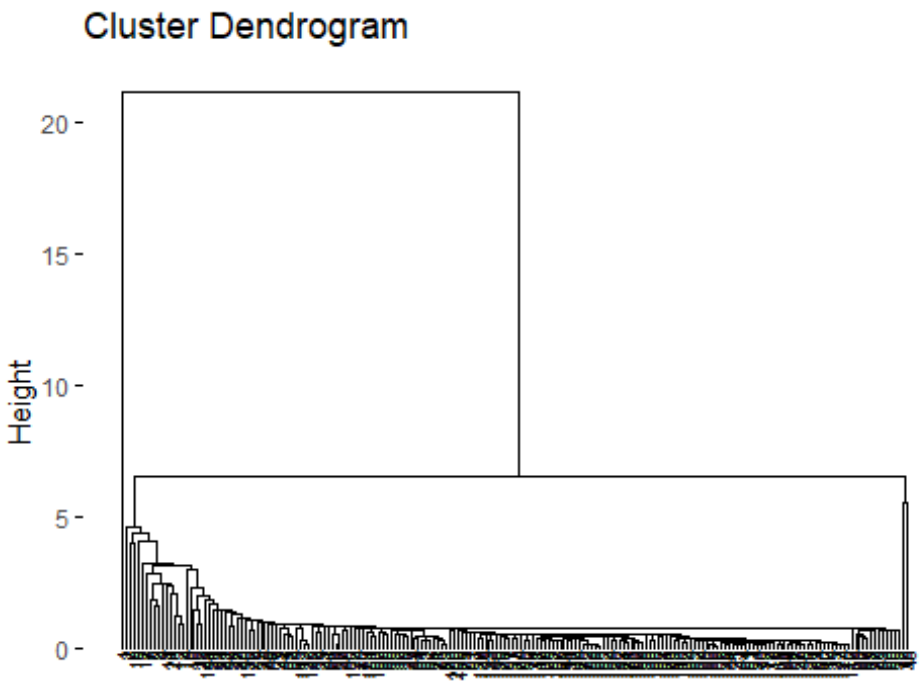
```
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 12 proposed  2 as the best number of clusters
## * 6 proposed  3 as the best number of clusters
## * 2 proposed  9 as the best number of clusters
## * 3 proposed  10 as the best number of clusters
## * 1 proposed  NA's as the best number of clusters
##
## Conclusion
```

```
## ************************
## * According to the majority rule, the best number of clusters is  2 .
```



According to the majority rule, the best number of clusters of this method is 2 .

```
slm.md1 <- hclust(dist.man.200, method = "single")
fviz_dend(slm.md1, cex = 0.5)
```



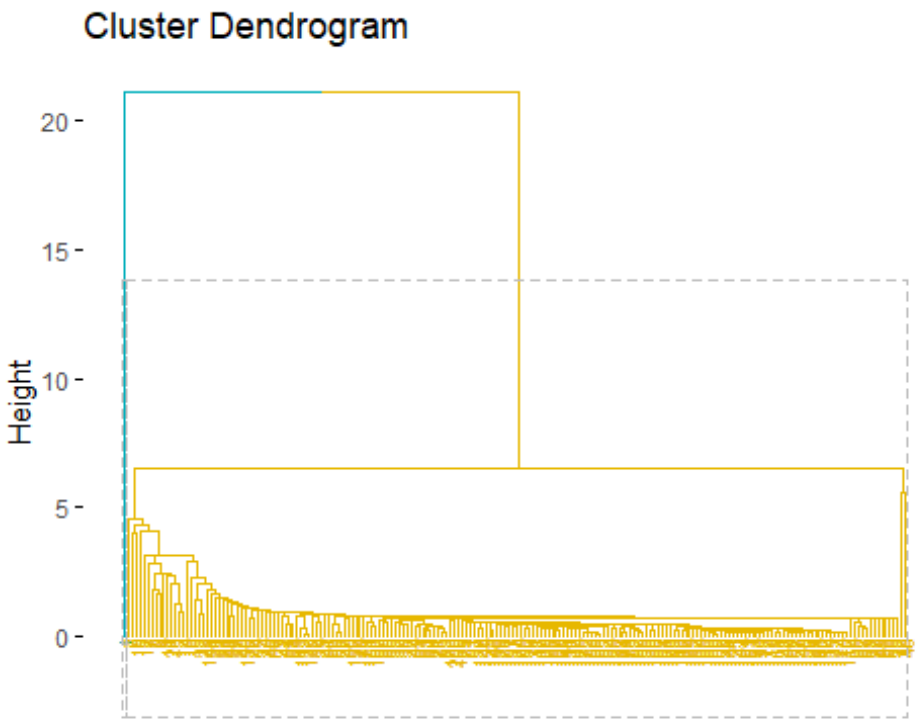
Looking at this dendogram is a little hard to visualize the partition of the clusters. So lets check the cophenetic results.

```
slm.md2 <- hclust(dist.man.200, method = "single")
cor(dist.man.200, cophenetic(slm.md2))

## [1] 0.9127879
```

The correlation coefficient of 0.91 shows that this clustering solution reflects our data and its a good one to use.

Now, to see the clusters more clearly, we have to cut the hierarchical tree:

```
slm.md3 <- cutree(slm.md2, k = 2)
fviz_dend(slm.md2, k = 2, cex = 0.5, k_colors = c("#00AFBB", "#E7B800"),
color_labels_by_k = TRUE, rect = TRUE)
```

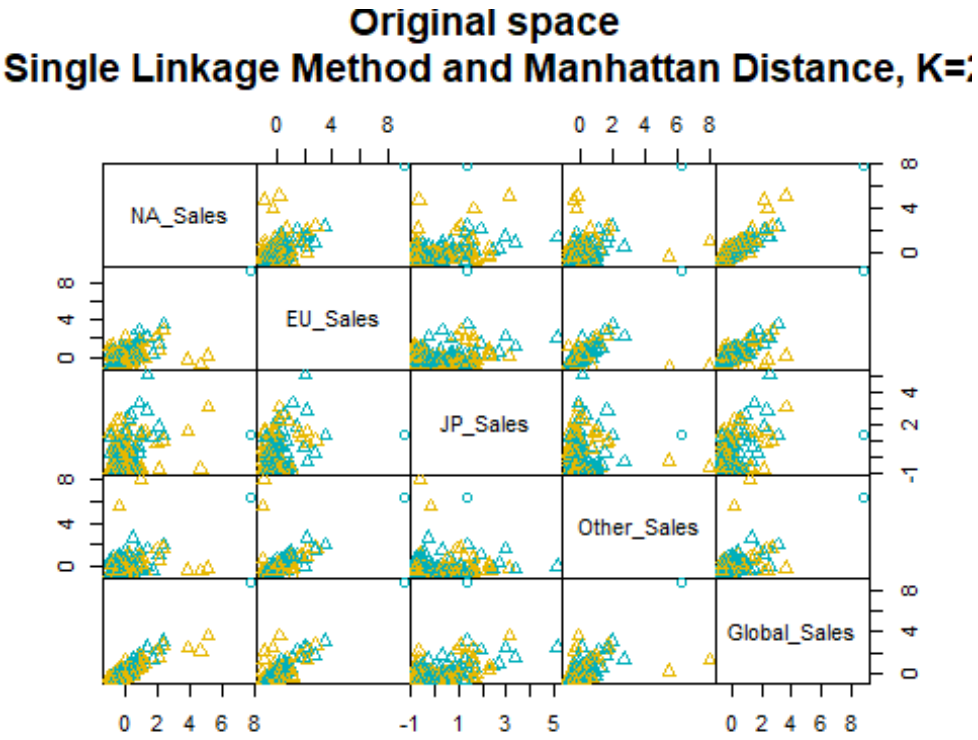



```
table(slm.md3)

## slm.md3
##   1   2
##   1 199
```

As we can see in the dendogram and in the table, the clusterization puts only 1 observation in the first cluster one more time.

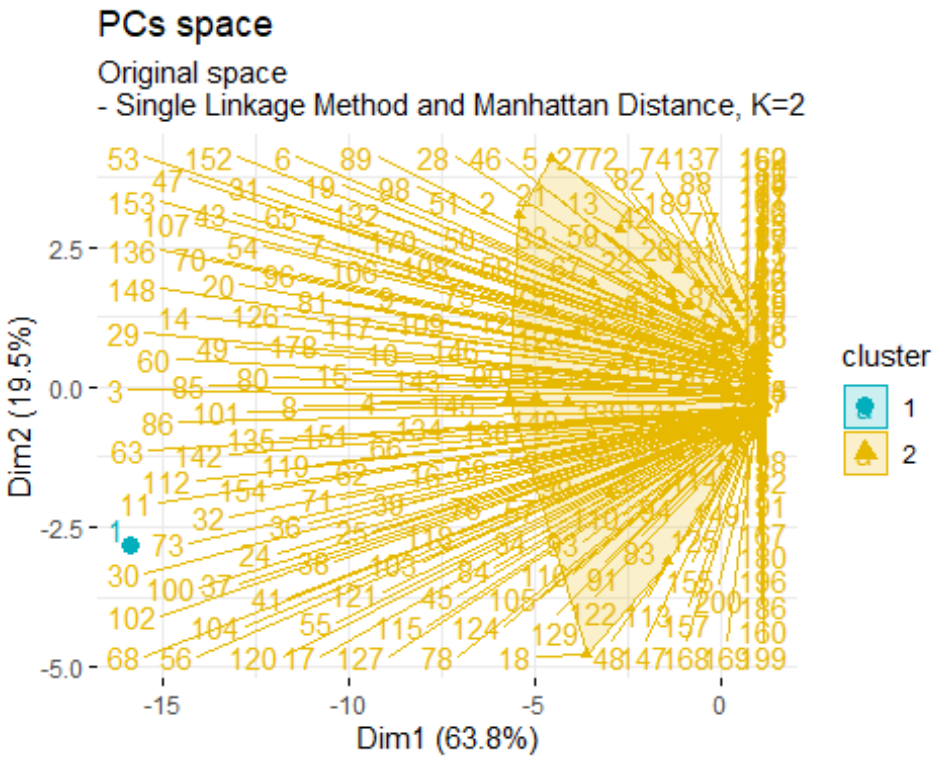
Now lets visualize the original space:

```
pairs(vg_sales.scaled, gap=0, pch=slm.md3, col=c("#00AFBB", "#E7B800"),
main="Original space\n- Single Linkage Method and Manhattan Distance, K=2
"[slm.md3])
```

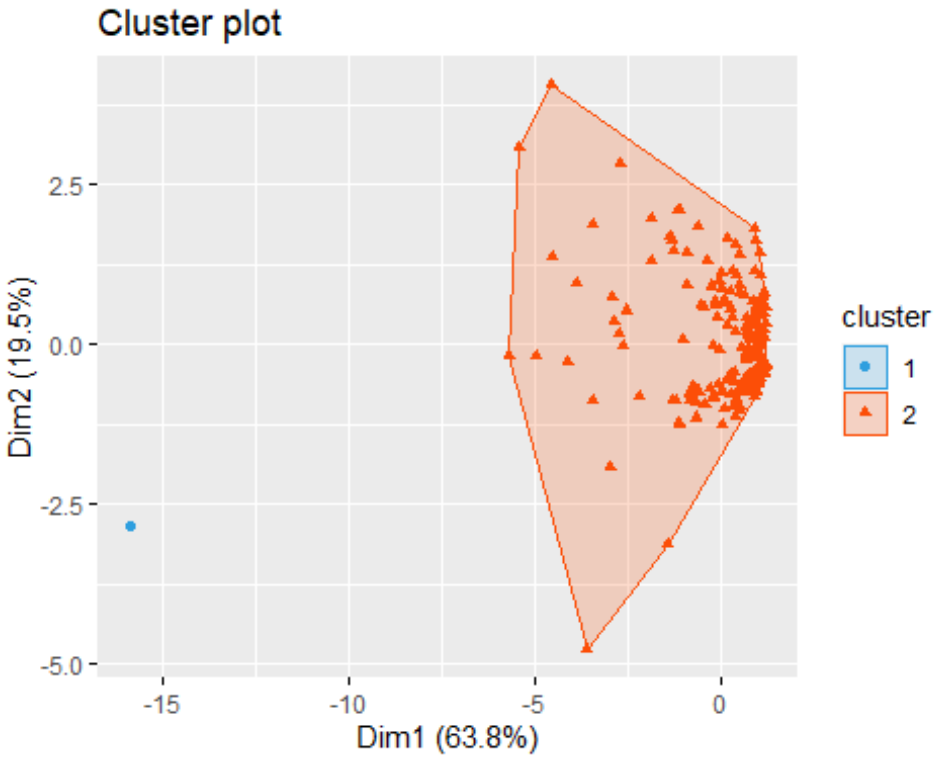


Using the function fviz *cluster()*, we can also visualize the results in the scatter plot of the first 2 PCs.

```
fviz_cluster(list(data = vg_sales.scaled, cluster = slm.md3), palette = c
("#00AFBB", "#E7B800"), ellipse.type = "convex", main="PCs space", repel
= TRUE,
show.clust.aver = FALSE, ggtheme = theme_minimal())+labs(
subtitle = "Original space\n- Single Linkage Method and Manhattan Distanc
e, K=2", cex.sub= 0.5)
```




```
fviz_cluster(list(data=vg_sales.scaled, cluster = slm.md3),geom="point",ellipse.type="convex",palette = c("#2E9FDF", "#FC4E07"),repel = TRUE, show.clust.cent = FALSE)
```



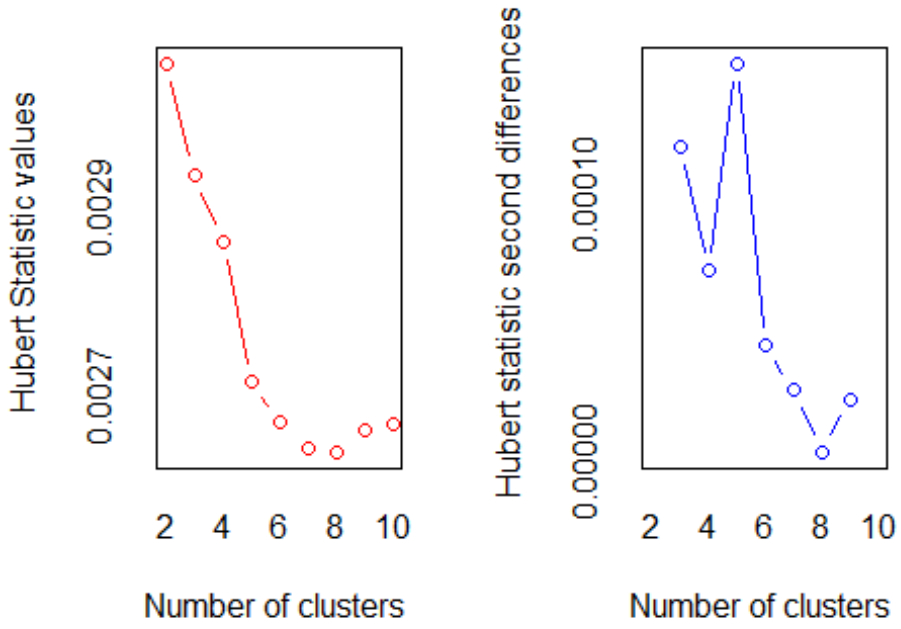
The single linkage method using the both distances gave us the same results of the average linkage method.

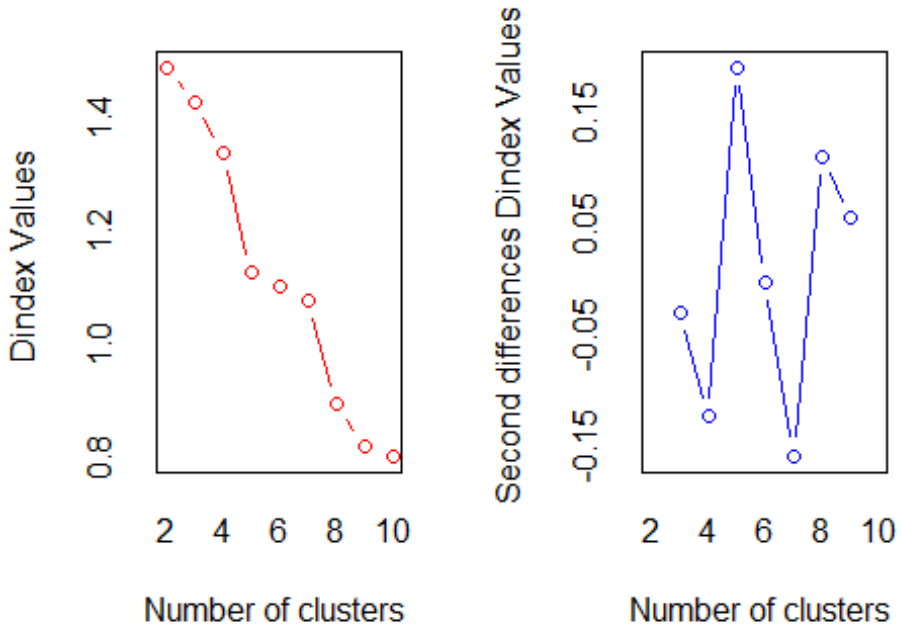
Complete Linkage Method and Euclidean Distance

The complete linkage method is a hierarchical classification method where the distance between two classes is defined as the greatest distance that could be obtained if we select one element from each class and measure the distance between these elements. In other words, it is the distance between the most distant elements from each class.

So lets use first the complete linkage method with the euclidean distance

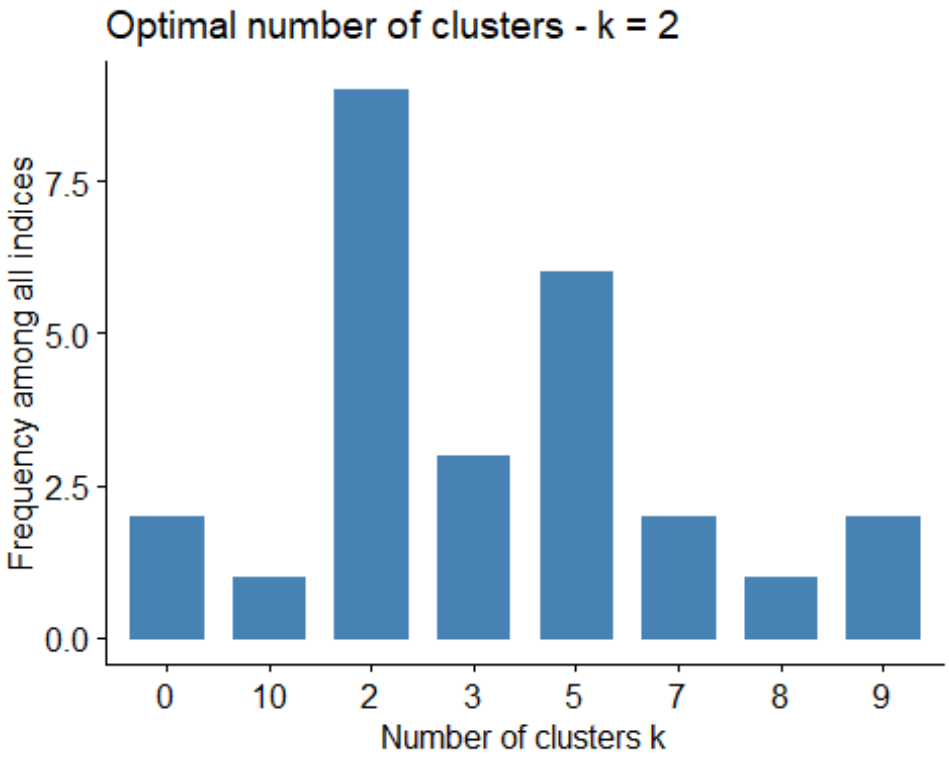
```
library(NbClust)
clm.ed <- NbClust(vg_sales.scaled, distance = "euclidean", min.nc = 2, ma
x.nc = 10,
method = "complete")
```





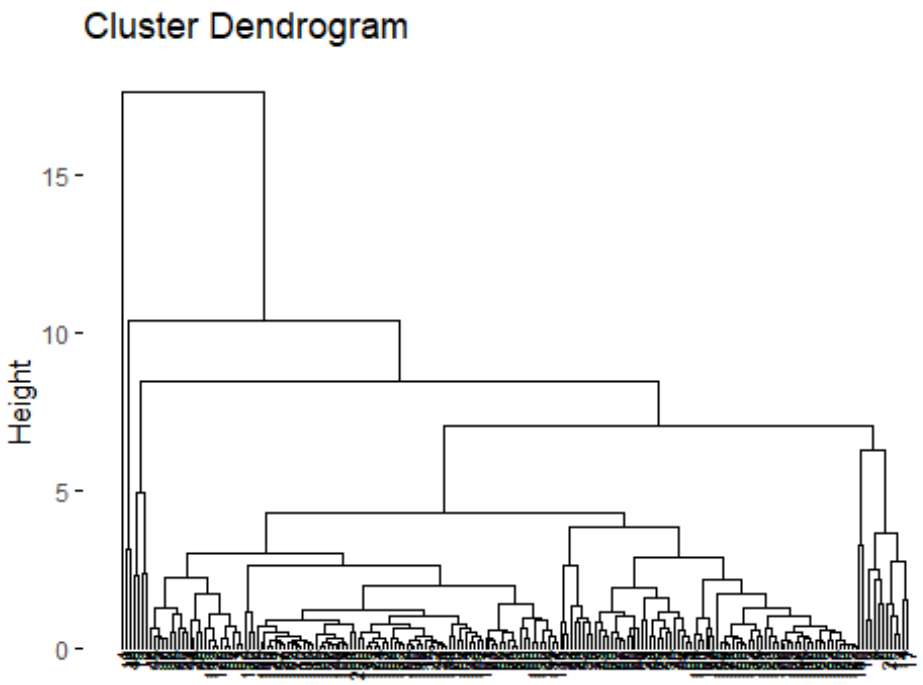

```
## *****************************************************************
## * Among all indices:
## * 9 proposed 2 as the best number of clusters
## * 3 proposed 3 as the best number of clusters
## * 6 proposed 5 as the best number of clusters
## * 2 proposed 7 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 2 proposed 9 as the best number of clusters
## * 1 proposed 10 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *****************************************************************

fviz_nbclust(clm.ed)

## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 9 proposed  2 as the best number of clusters
## * 3 proposed  3 as the best number of clusters
## * 6 proposed  5 as the best number of clusters
## * 2 proposed  7 as the best number of clusters
```

```
## * 1 proposed  8 as the best number of clusters
## * 2 proposed  9 as the best number of clusters
## * 1 proposed  10 as the best number of clusters
##
## Conclusion
## =========================
## * According to the majority rule, the best number of clusters is  2 .
```



According to the majority rule, the best number of clusters for this method is 2 .

```
clm.ed1 <- hclust(dist.eucl.200, method = "complete")
fviz_dend(clm.ed1, cex = 0.5)
```



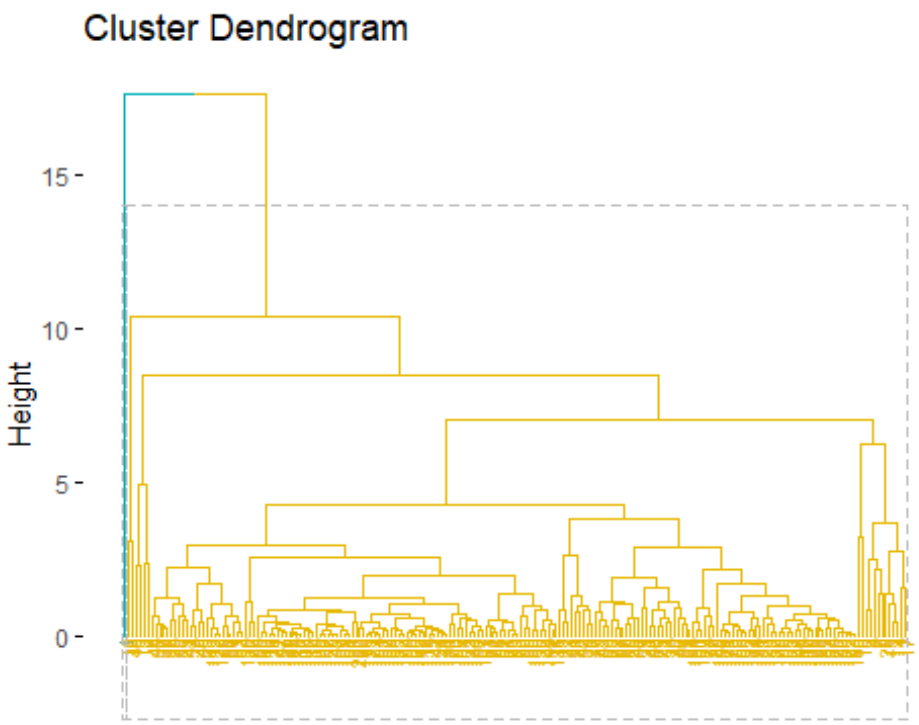
Now lets check the correlation between the cophenetic distance and the original one:

```
clm.ed2 <- hclust(dist.eucl.200, method = "complete")
cor(dist.eucl.200, cophenetic(clm.ed2))

## [1] 0.8948669
```

The correlation coefficient of 0.89 shows that this clustering solution reflects our data and its a good one to use.

Now, if we want to see clusters, we have to cut the hierarchical tree and specify the number of clusters that we want:

```
clm.ed3 <- cutree(clm.ed2, k = 2)
fviz_dend(clm.ed2, k = 2, cex = 0.5, k_colors = c("#00AFBB", "#E7B800"),
color_labels_by_k = TRUE, rect = TRUE)
```

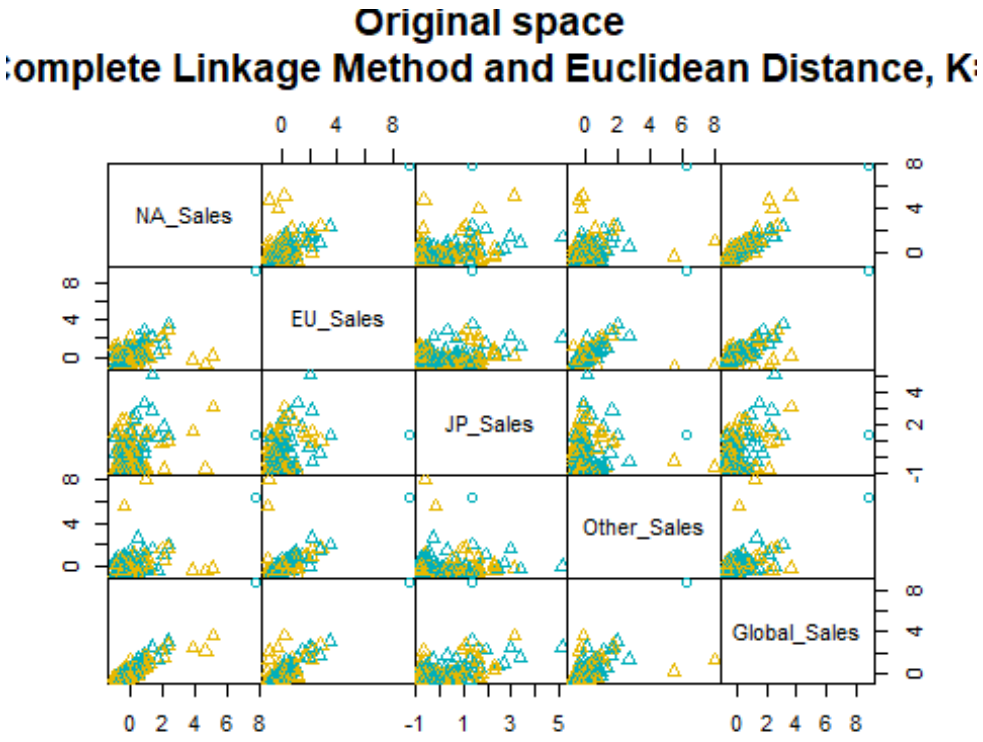



```
table(clm.ed3)

## clm.ed3
##   1   2
##   1 199
```

As we can see in the dendogram and in the table, the clusterization puts only 1 observation in the first cluster, exactly the way the average linkage method and the single linkage method did.

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

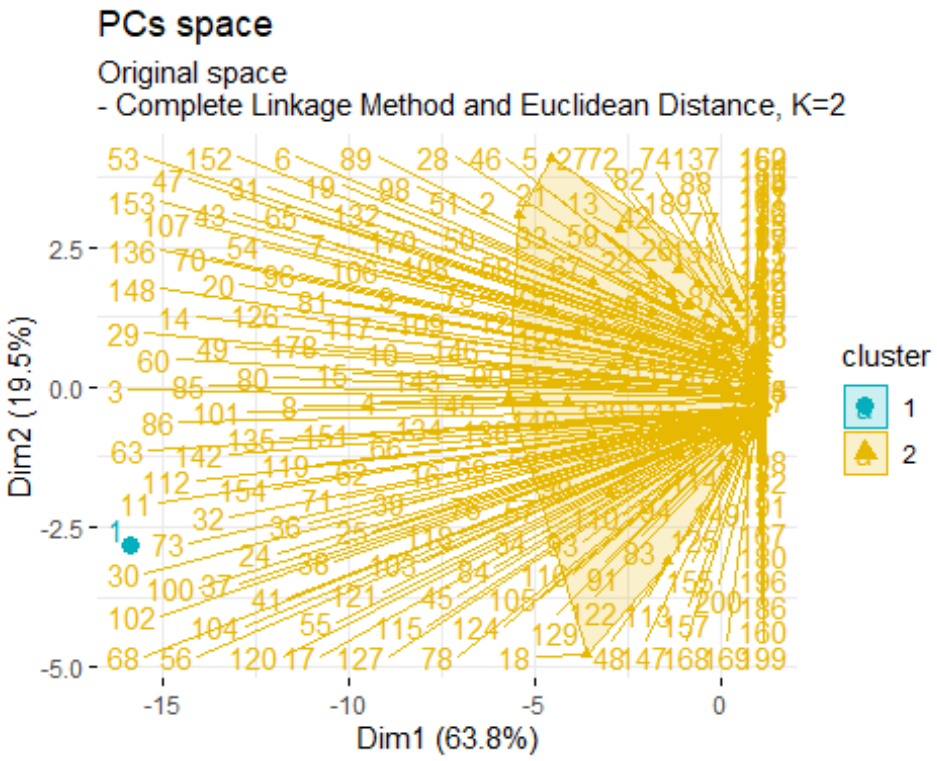
```
pairs(vg_sales.scaled, gap=0, pch=clm.ed3, col=c("#00AFBB", "#E7B800"),
main="Original space\n- Complete Linkage Method and Euclidean Distance, K
=2"[clm.ed3])
```



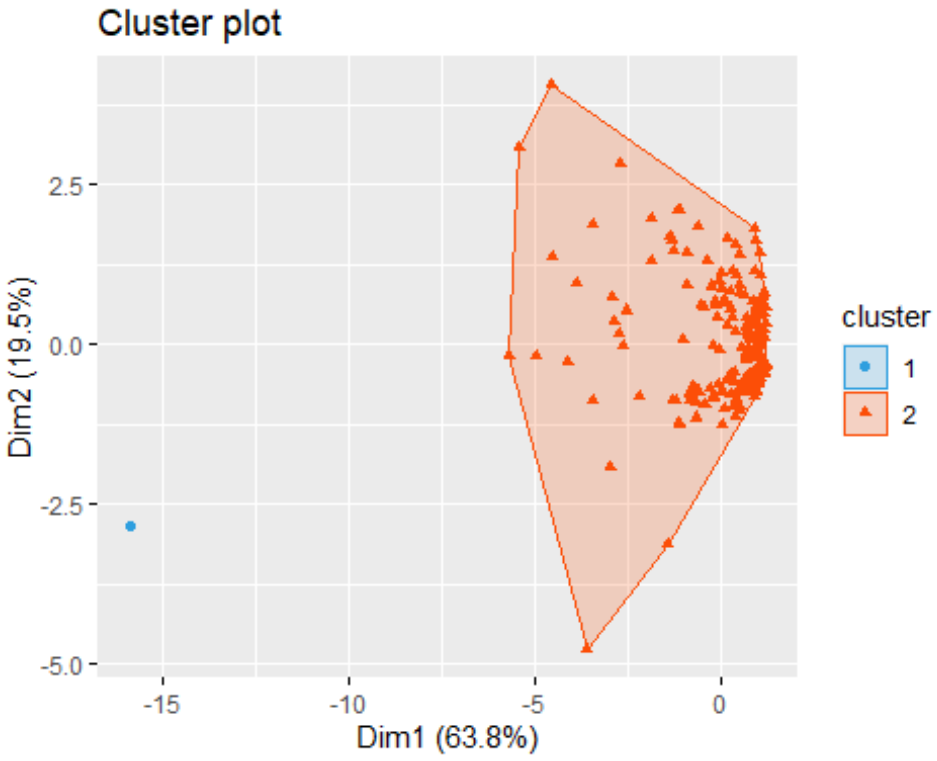
This original space is much alike the other ones and we see a lot of overlaping.

Now lets visualize the results in the scatter plot of the first 2 PCs.

```
fviz_cluster(list(data = vg_sales.scaled, cluster = clm.ed3), palette = c
("#00AFBB", "#E7B800"), ellipse.type = "convex", main="PCs space", repel
= TRUE,
show.clust.aver = FALSE, ggtheme = theme_minimal())+labs(
subtitle = "Original space\n- Complete Linkage Method and Euclidean Dista
nce, K=2", cex.sub= 0.5)
```




```
fviz_cluster(list(data=vg_sales.scaled, cluster = clm.ed3),geom="point",ellipse.type="convex",palette = c("#2E9FDF", "#FC4E07"),repel = TRUE, show.clust.cent = FALSE)
```

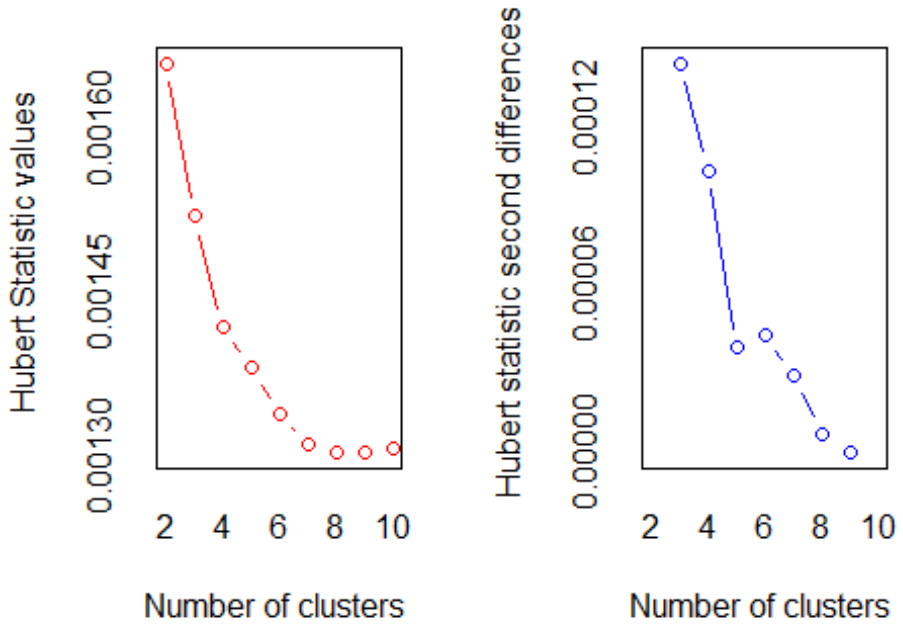


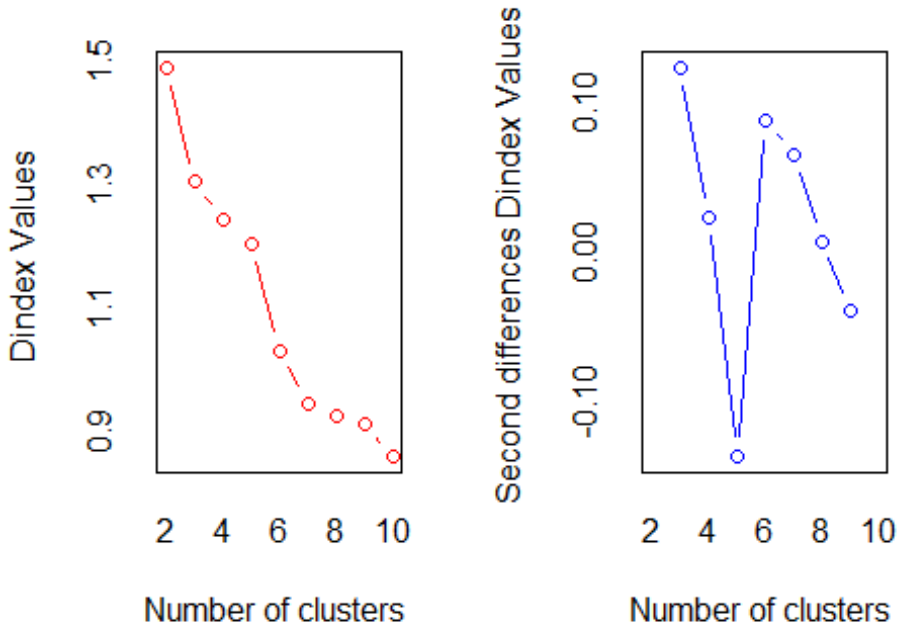
The results of the complete linkage method using euclidean distance is the same as the average and single linkage method.

Complete Linkage Method and Manhattan Distance

Now lets use the Manhattan distance to analyse the complete linkage Method:

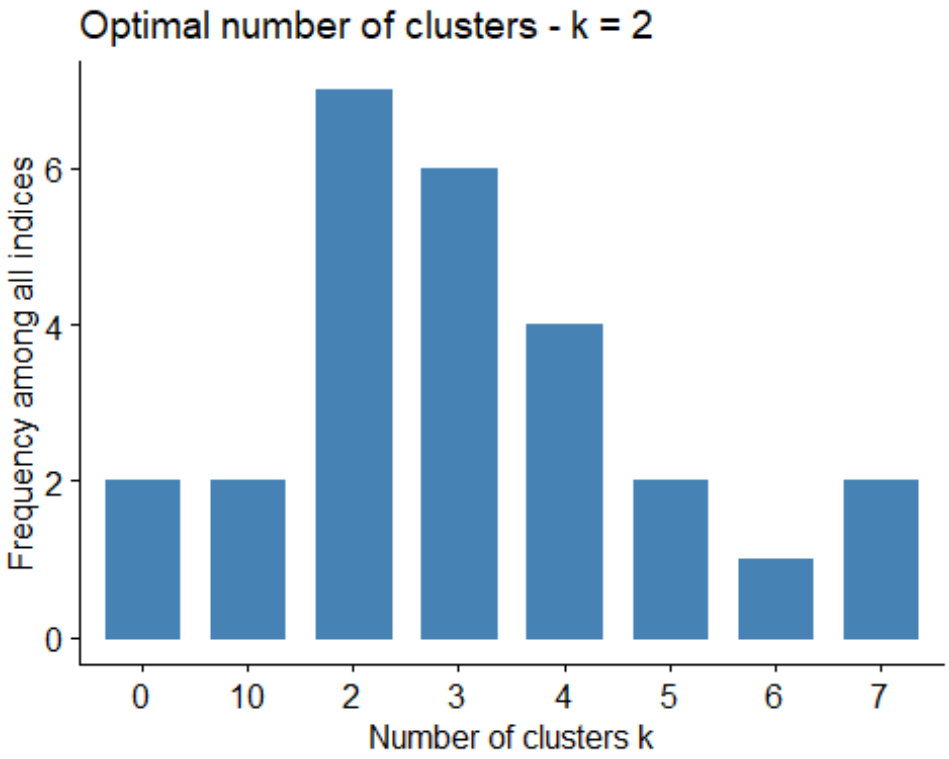
```
library(NbClust)
clm.md <- NbClust(vg_sales.scaled, distance = "manhattan", min.nc = 2, ma
x.nc = 10,
method = "complete")
```





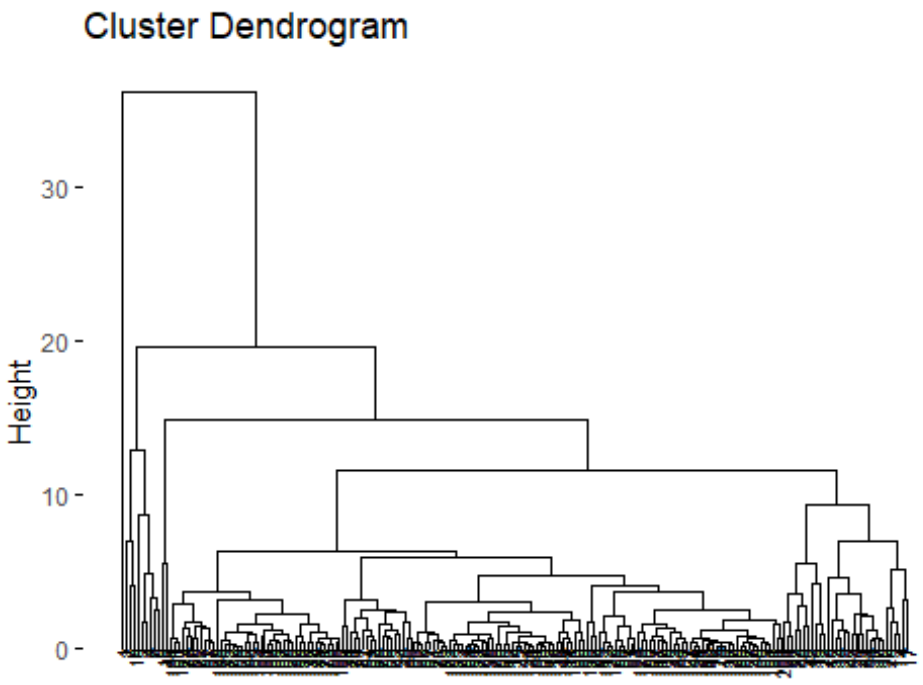

```
## *****************************************************************
## * Among all indices:
## * 7 proposed 2 as the best number of clusters
## * 6 proposed 3 as the best number of clusters
## * 4 proposed 4 as the best number of clusters
## * 2 proposed 5 as the best number of clusters
## * 1 proposed 6 as the best number of clusters
## * 2 proposed 7 as the best number of clusters
## * 2 proposed 10 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *****************************************************************

fviz_nbclust(clm.md)

## Among all indices:
## ==================
## * 2 proposed  0 as the best number of clusters
## * 7 proposed  2 as the best number of clusters
## * 6 proposed  3 as the best number of clusters
## * 4 proposed  4 as the best number of clusters
## * 2 proposed  5 as the best number of clusters
```

```
## * 1 proposed  6 as the best number of clusters
## * 2 proposed  7 as the best number of clusters
## * 2 proposed  10 as the best number of clusters
##
## Conclusion
## =========================
## * According to the majority rule, the best number of clusters is  2 .
```



According to the majority rule, the best number of clusters is 2 .

```
clm.md1 <- hclust(dist.man.200, method = "complete")
fviz_dend(clm.md1, cex = 0.5)
```



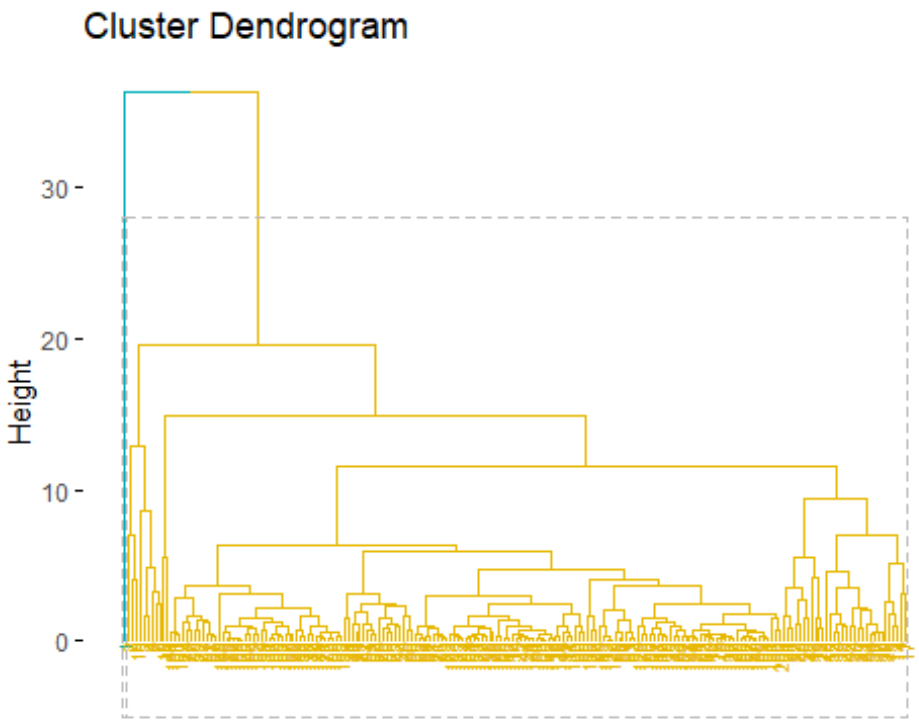
Lets check the correlation between the cophenetic distance and the original one:

```
clm.md2 <- hclust(dist.man.200, method = "complete")
cor(dist.man.200, cophenetic(clm.md2))

## [1] 0.8879684
```

The correlation coefficient shows that this clustering solution reflects our data and its a good one to use.

Now, if we want to see clusters, we have to cut the hierarchical tree:

```
clm.md3 <- cutree(clm.md2, k = 2)
fviz_dend(clm.md2, k = 2, cex = 0.5, k_colors = c("#00AFBB", "#E7B800"),
color_labels_by_k = TRUE, rect = TRUE)
```

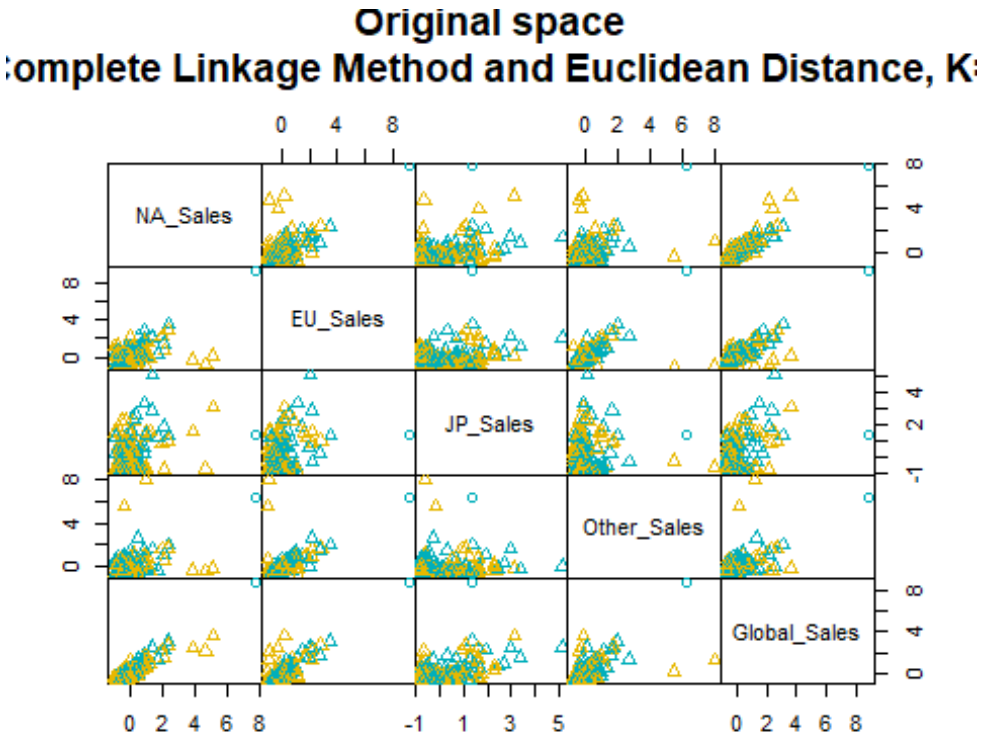



```
table(clm.md3)

## clm.md3
##   1   2
##   1 199
```

As we can see in the dendogram and in the table, the clusterization puts only 1 observation in the first cluster.

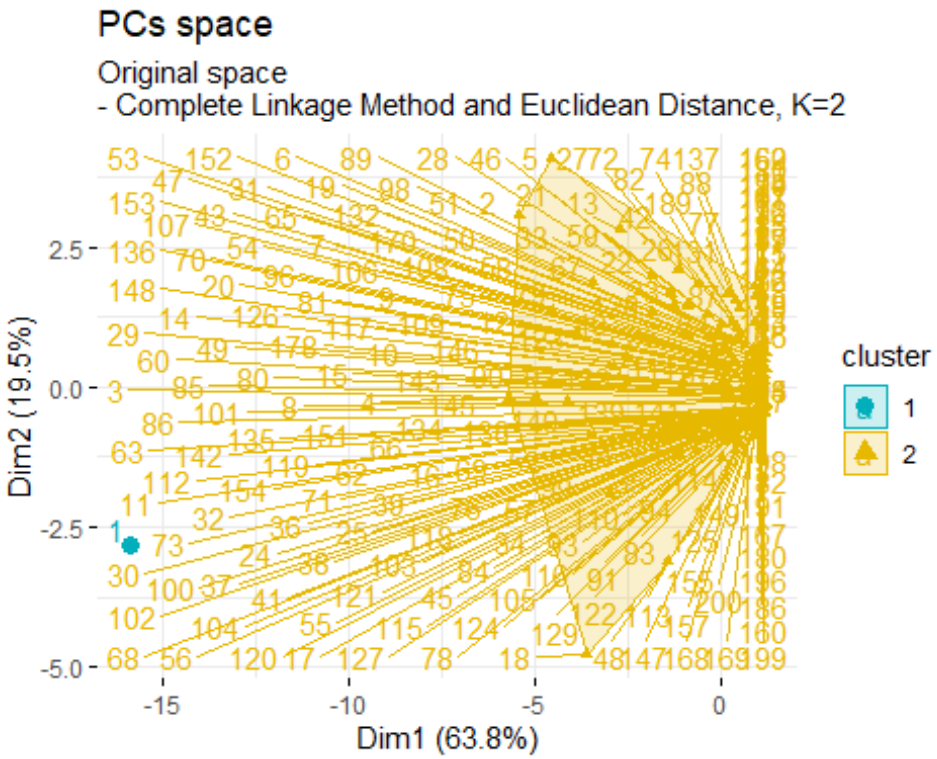
We can visualize these clustering results in the original space:

```
pairs(vg_sales.scaled, gap=0, pch=clm.md3, col=c("#00AFBB", "#E7B800"),
main="Original space\n- Complete Linkage Method and Euclidean Distance, K
=2"[clm.md3])
```

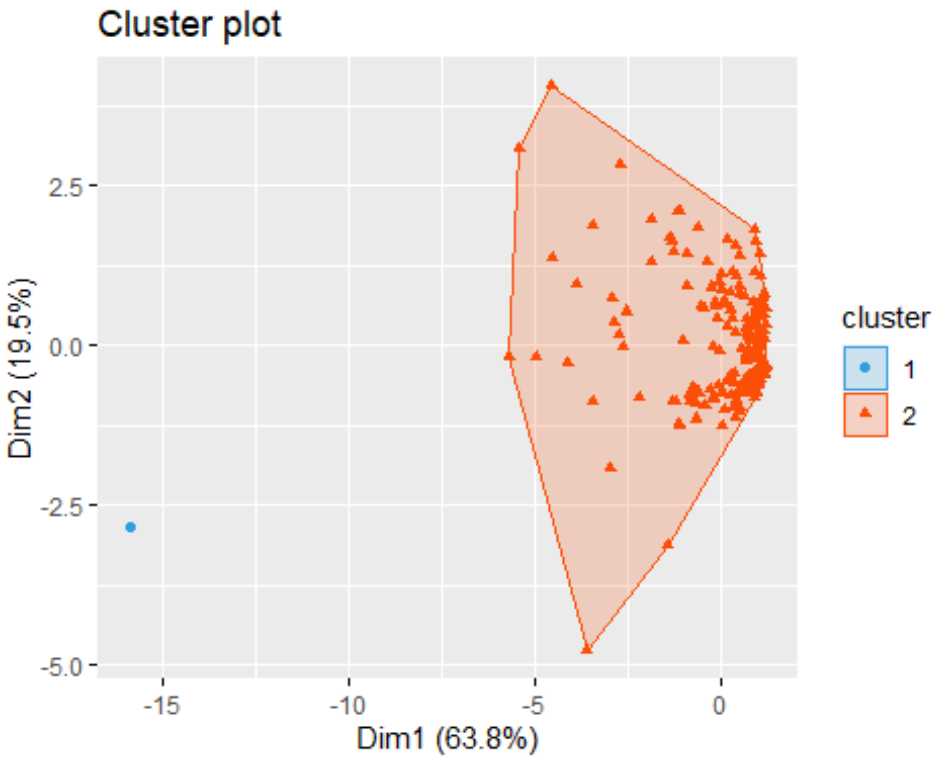


Now we can also visualize the results in the scatter plot of the first 2 PCs.

```
fviz_cluster(list(data = vg_sales.scaled, cluster = clm.md3), palette = c
("#00AFBB", "#E7B800"), ellipse.type = "convex", main="PCs space", repel
= TRUE,
show.clust.aver = FALSE, ggtheme = theme_minimal())+labs(
subtitle = "Original space\n- Complete Linkage Method and Euclidean Dista
nce, K=2", cex.sub= 0.5)
```




```
fviz_cluster(list(data=vg_sales.scaled, cluster = clm.md3),geom="point",ellipse.type="convex",palette = c("#2E9FDF", "#FC4E07"),repel = TRUE, show.clust.cent = FALSE)
```



As we can see, we had the same result of partitioning the datas in 2 clusters and put the number 1 on his own cluster.

Hierarchical clustering - Partial conclusion

After trying these many methods and combine them with the type of distances, in first instance I would choose the combination of the Ward's Method with the Manhattan distance, even if he didnt get the minimun point of the cophenetic coefficient (0.75) but it gets really close with 0.72 and it was one of the few combinations which brought us more than only 2 clusters. Even if the 1st cluster of this method has one observation, it would be the one I would choose.

This one single cluster with only one observation (the number one in the rank itself) shows very clearly that he was an outlier and that all the methods so far were sensible to outliers.

Partitioning Clustering

In the partitioning clustering, the number of clusters K is decided before to call the method. Once the number of clusters is specified, the data are split into clusters in such a way that the within-cluster dissimilarity is minimized.

We will use 2 types of partitioning methods:

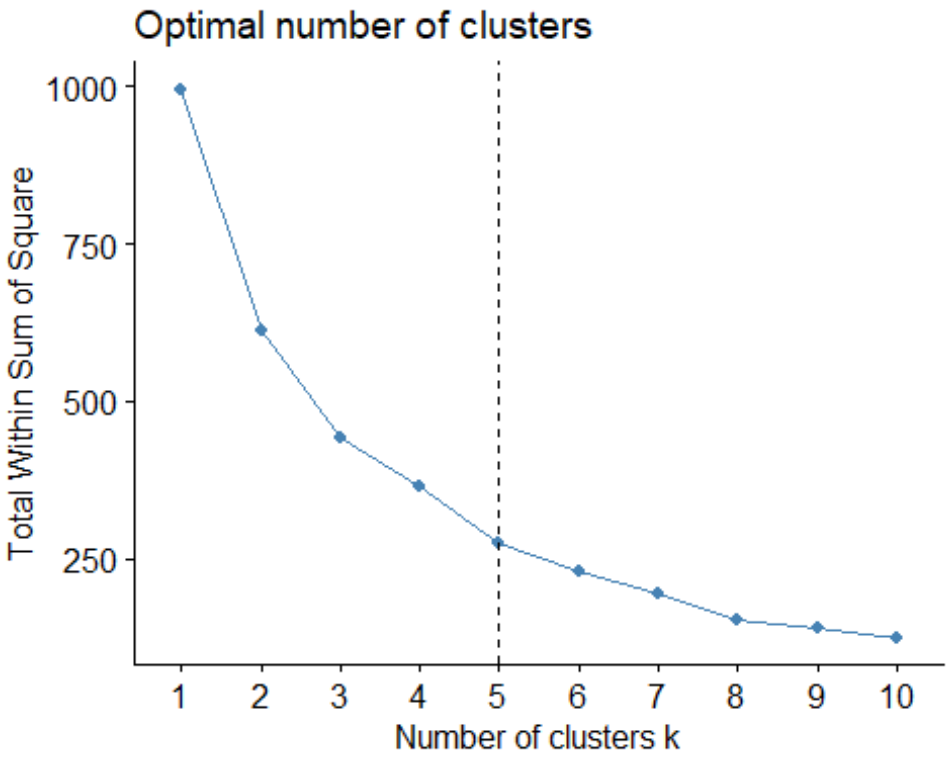
- K-Means: is a Clustering method that aims to partition *n* observations among *k* groups where each observation belongs to the group closest to the average. It is a method that is senstive to outliers.
- K-Medoids: is based on the medoids (which is a point that belongs to the data set) calculated by minimizing the absolute distance between the points and the selected centroid, instead of minimizing the square distance. Is also known as partitioning around medoids (PAM) and is less senstive to outliers when compared to the k-means.

K-Means

Estimating the Optimal Number of Clusters

The K-means clustering requires the users to specify the number of clusters K to be generated, so we will use the *Elbow Methods* to determine a good number of *K* to work on:

```
library(factoextra)
fviz_nbclust(vg_sales.scaled, kmeans, nstart = 25, method = "wss")+
   geom_vline( xintercept = 5, linetype = 2)
```



The plot above represents WSS as a function of *K*. WSS decreases as K increases, but it can be seen a bend (or "elbow") at K = 5. This bend indicates that additional clusters beyond the fifith have a little improvement in terms of WSS.

So we will start the K-means methods by radomly selecting the centroids using the *set.seed()* function

```
set.seed(123)
km.res <- kmeans(vg_sales.scaled, 5, nstart = 25)
print(km.res)
```

```
## K-means clustering with 5 clusters of sizes 2, 19, 1, 126, 52
##
## Cluster means:
##     NA_Sales   EU_Sales   JP_Sales Other_Sales Global_Sales
## 1  0.2809896 -1.0098415 -0.3831914   6.7866585    0.7198118
## 2  1.7921678  1.4943435  1.5436013   0.8078679    1.9727400
## 3  7.7414716  9.2249668  1.3764948   6.2929404    8.7728841
## 4 -0.2353996 -0.1561445 -0.5938590  -0.1159653   -0.3279987
## 5 -0.2441208 -0.3062232  0.8632250  -0.3962330   -0.1224371
##
## Clustering vector:
##   [1] 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 5 5 4 4 5 5 5 5 4 5 4
5 4 4 4 4
##  [38] 4 4 5 4 5 5 4 4 5 5 1 4 5 5 4 5 5 4 4 4 5 5 4 4 4 4 5 5 4 5 4 4
5 4 5 4 5
##  [75] 5 4 5 4 5 4 4 5 4 4 4 4 5 5 5 4 4 4 4 4 4 4 4 5 4 4 4 4 4 4 4 4
```

```
4 4 4 4 4
## [112] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 4 4 5 5 4 4 4 4 5 4 4 4 4 4 4
4 4 4 4 5
## [149] 4 5 4 5 4 4 4 5 4 5 4 4 4 5 4 4 4 4 4 4 4 5 5 4 4 5 5 4 4 4 4 4
5 4 4 4 5
## [186] 4 4 5 5 4 4 5 4 4 4 4 4 4 4 4
##
## Within cluster sum of squares by cluster:
## [1]   4.862275 122.011526   0.000000  92.439410  56.077391
##  (between_SS / total_SS =  72.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.w
ithinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

We can see in the results above that we have with k-means 5 clusters of sizes 9, 50, 1, 125, 15, and it also indicates the position of the video game it respective cluster.

Now lets compute the mean of each variable by cluster, using the original data, via the *aggregate()* function:

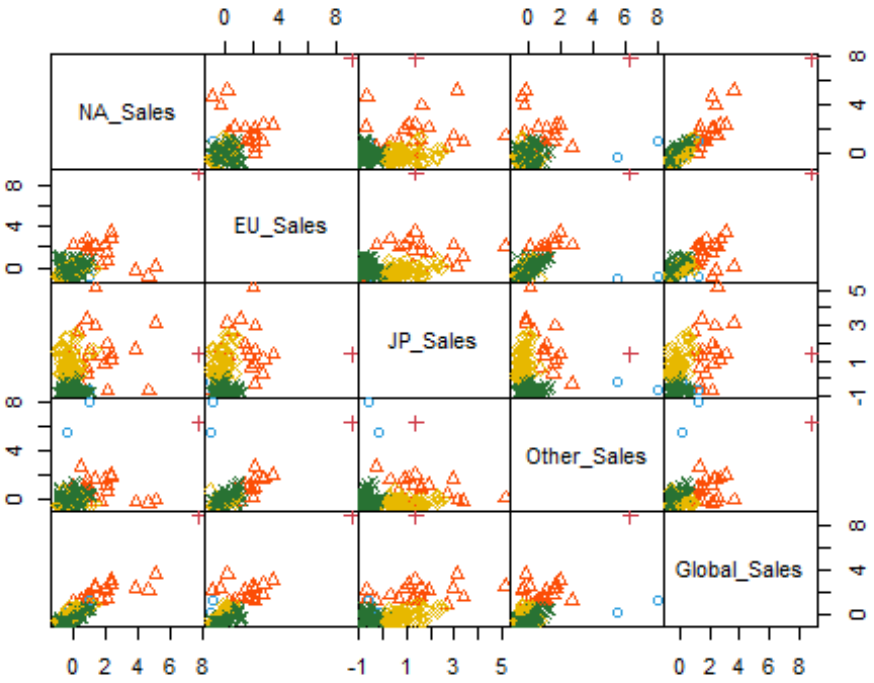
```
aggregate(vg_sales.scaled, by=list(cluster = km.res$cluster), mean)
```

```
##   cluster   NA_Sales   EU_Sales   JP_Sales Other_Sales Global_Sales
## 1       1  0.2809896 -1.0098415 -0.3831914   6.7866585    0.7198118
## 2       2  1.7921678  1.4943435  1.5436013   0.8078679    1.9727400
## 3       3  7.7414716  9.2249668  1.3764948   6.2929404    8.7728841
## 4       4 -0.2353996 -0.1561445 -0.5938590  -0.1159653   -0.3279987
## 5       5 -0.2441208 -0.3062232  0.8632250  -0.3962330   -0.1224371
```

Based on this we can see that the third cluster contains higher values when compared to the other and the fourth cluster contains the smallest values.

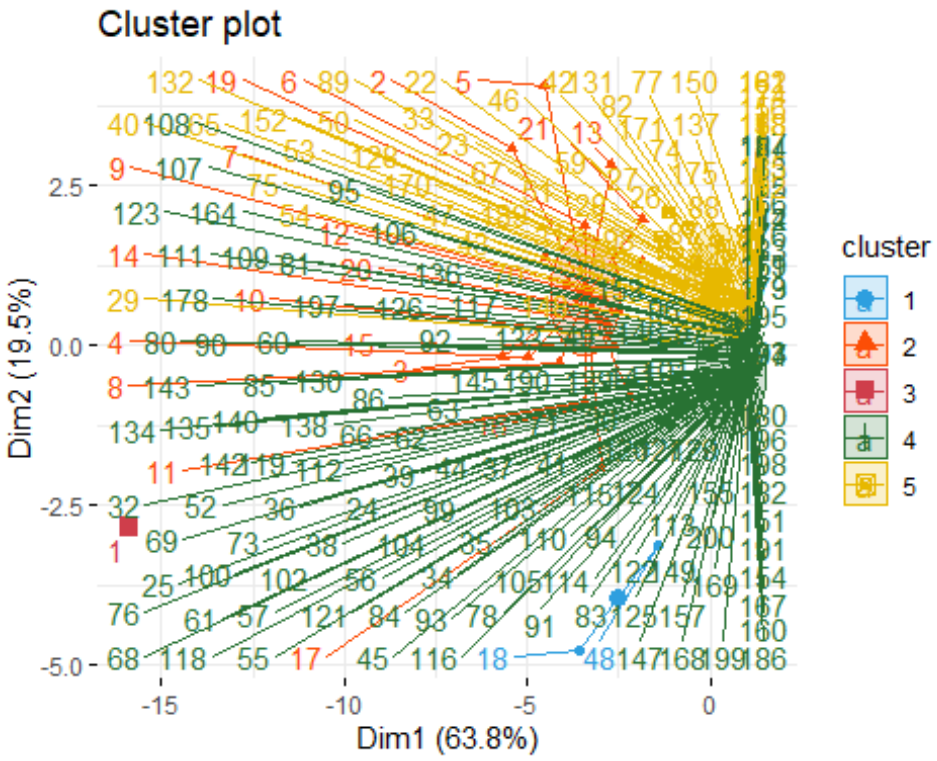
Now lets visualize it in the original space:

```
cl.km <- km.res$cluster
pairs(vg_sales.scaled, gap=0, pch=cl.km, col = c("#2E9FDF", "#FC4E07", "#
CF3E4B", "#287233", "#E7B800")[cl.km])
```



Now lets visualize the k means cluster in a low dimensional space:

```
fviz_cluster(km.res,
             data = vg_sales.scaled,
             palette = c("#2E9FDF", "#FC4E07", "#CF3E4B", "#287233", "#E7
B800"),
             ellipse.type = "euclid",
             star.plot = TRUE,
             repel = TRUE,
             ggtheme = theme_minimal())
```



The issue with this method is that the final results obtained is sensitive to the initial random selection of cluster centers. Which means that for every different run of the algorithm on the same data set, you may choose a different set of initial centers. This may lead to different clustering results on different runs of the algorithm and also, as we can verify, the K-means is sensitive to outliers.

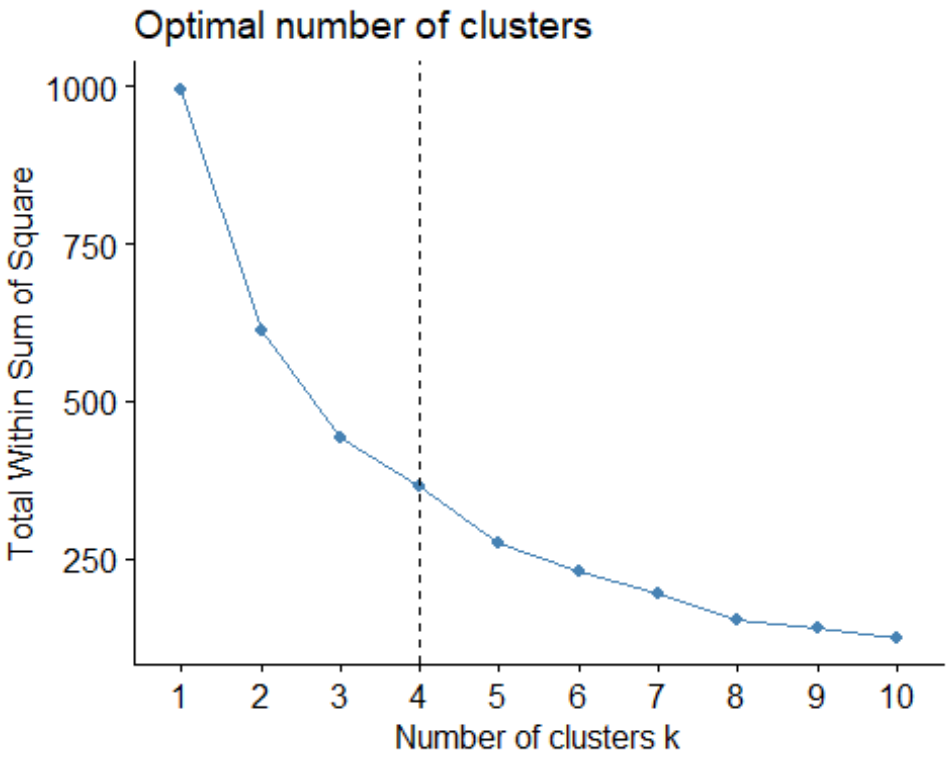
And besides of that, we may choose the wrong *k* in the beginning of the analysis.

To try dicrease some of these issues, we can 2 two things:

- Compute the K-means for a different range of *K* valuews and compare their results.
- To avoid the distortions caused by outliers we can use the PAM algorithm, also knows as the K-medoids, that we will do in the next topic.

I will do the method again with only one different *K* just to check:

```
library(factoextra)
fviz_nbclust(vg_sales.scaled, kmeans, nstart = 25, method = "wss")+
   geom_vline( xintercept = 4, linetype = 2)
```




```
set.seed(123)
km.res1 <- kmeans(vg_sales[,6:10], 4, nstart = 25)
print(km.res1)
```

```
## K-means clustering with 4 clusters of sizes 16, 48, 1, 135
##
## Cluster means:
##    NA_Sales  EU_Sales  JP_Sales Other_Sales Global_Sales
## 1 14.733750  7.175000 4.1187500   1.7900000    27.817500
## 2  6.170625  3.852083 1.8522917   1.4743750    13.348542
## 3 41.490000 29.020000 3.7700000   8.4600000    82.740000
## 4  2.999259  2.080741 0.9165185   0.5933333     6.589333
##
## Clustering vector:
##   [1] 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 4 4 4 4 4 4 4 4
##  [75] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [112] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [149] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [186] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##
## Within cluster sum of squares by cluster:
```

```
## [1] 1368.9047 1042.0241    0.0000  773.0796
##  (between_SS / total_SS =  84.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.w
ithinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

As we can check, now the clusters are 4 and it contains 16,48, 1 and 135 observations each.

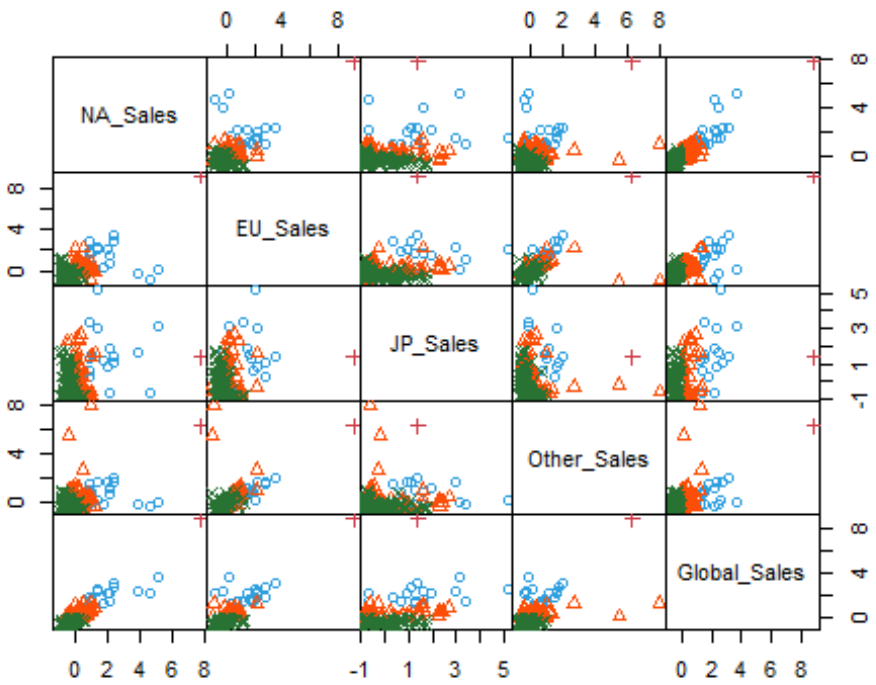
```
aggregate(vg_sales[,6:10], by=list(cluster = km.res1$cluster), mean)
```

```
##   cluster  NA_Sales  EU_Sales  JP_Sales Other_Sales Global_Sales
## 1       1 14.733750  7.175000 4.1187500   1.7900000    27.817500
## 2       2  6.170625  3.852083 1.8522917   1.4743750    13.348542
## 3       3 41.490000 29.020000 3.7700000   8.4600000    82.740000
## 4       4  2.999259  2.080741 0.9165185   0.5933333     6.589333
```

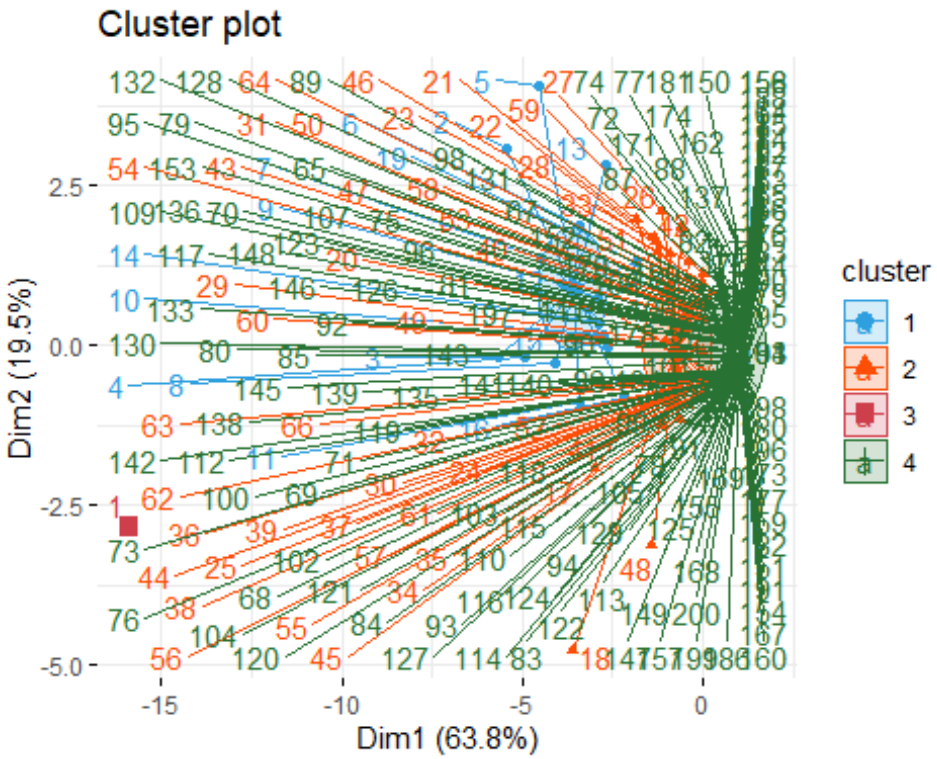
With this result we see that the third cluster contains much higher values than the other ones.

Now lets visualize the datas in the original space

```
cl.km1 <- km.res1$cluster
pairs(vg_sales.scaled, gap=0, pch=cl.km1, col = c("#2E9FDF", "#FC4E07", "
#CF3E4B", "#287233")[cl.km1])
```



```
fviz_cluster(km.res1,
             data = vg_sales.scaled,
             palette = c("#2E9FDF", "#FC4E07", "#CF3E4B", "#287233"),
             ellipse.type = "euclid",
             star.plot = TRUE,
             repel = TRUE,
             ggtheme = theme_minimal())
```



The SS value of the first method with 5 clusters is 88% while the second with 4 is 84% which means that the first one with k=5 is better.

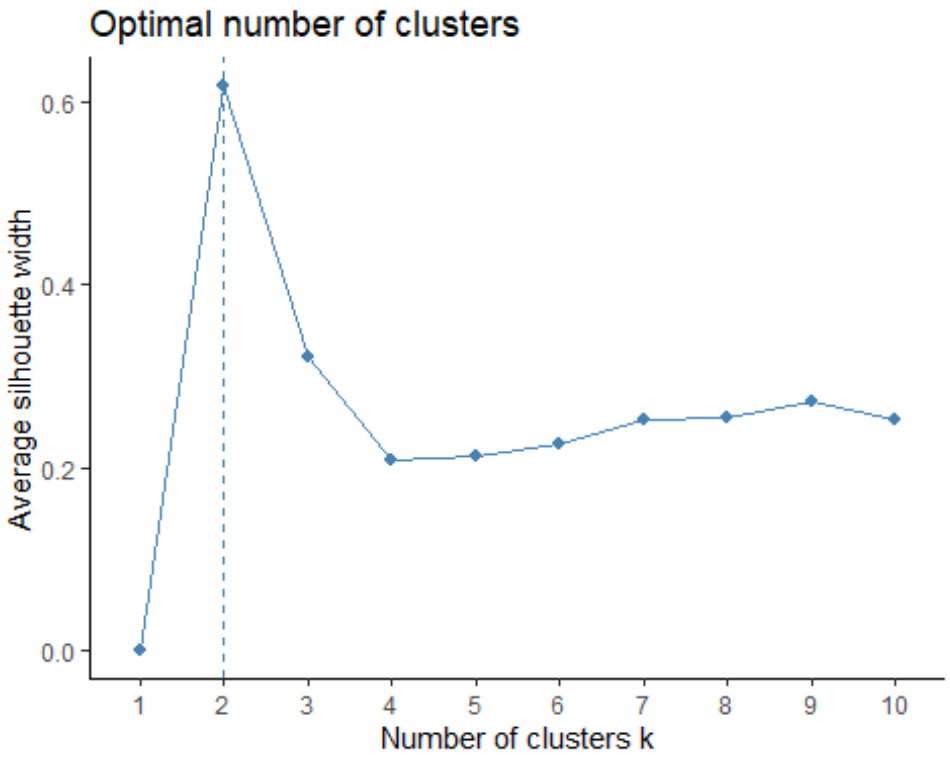
K-medoids (PAM)

Estimating the Optimal Number of Clusters

The K-medoids algorithm requires the user to specify *K*, like in the k-means method. A good approach to determine the optimal number of clusters is the silhouette method.

The silhouette value, for each unit, is a measure of how similar a unit is to its own cluster compared to other clusters.

```
library(cluster)
library(factoextra)
fviz_nbclust(vg_sales.scaled, pam, method = "silhouette")+
  theme_classic()
```



From the plot, the suggested number of clusters is K = 2.

Now lets compute the PAM algorithm with *k=5* to see if can avoid the distortion caused by the outliers;

```
pam.res <- pam(vg_sales.scaled, 2, metric = "euclidean", stand = FALSE)
summary(pam.res)
```

```
## Medoids:
##       ID   NA_Sales   EU_Sales   JP_Sales Other_Sales Global_Sales
## [1,]  12  1.0403639  1.6061350  1.5866066   0.8201998     1.589842
## [2,] 101 -0.2245548 -0.3047888 -0.4736562  -0.2007090    -0.360914
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 2 2 2
1 2 2 2 2
##  [38] 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2
## [112] 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
## [149] 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
## [186] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## Objective function:
##    build     swap
## 1.236370 1.231602
##
```

```
## Numerical information per cluster:
##      size  max_diss   av_diss  diameter separation
## [1,]   28 13.584645 2.9074446 16.240642  0.6088598
## [2,]  172  5.801572 0.9587904  6.765931  0.6088598
##
## Isolated clusters:
##  L-clusters: character(0)
##  L*-clusters: character(0)
##
## Silhouette plot information:
##     cluster neighbor   sil_width
## 7         1        2  0.32528374
## 3         1        2  0.30905096
## 4         1        2  0.30654918
## 9         1        2  0.30452023
## 8         1        2  0.26896552
## 5         1        2  0.25964423
## 2         1        2  0.23921186
## 12        1        2  0.19293475
## 13        1        2  0.19216988
## 1         1        2  0.16722926
## 6         1        2  0.16041275
## 14        1        2  0.14889694
## 11        1        2  0.12692769
## 20        1        2  0.06200614
## 15        1        2  0.03693993
## 21        1        2  0.03003703
## 10        1        2 -0.01454536
## 18        1        2 -0.01693568
## 17        1        2 -0.02415215
## 19        1        2 -0.08857118
## 28        1        2 -0.15001405
## 26        1        2 -0.16110699
## 27        1        2 -0.16183785
## 22        1        2 -0.17655056
## 16        1        2 -0.17740029
## 23        1        2 -0.24270562
## 42        1        2 -0.30038393
## 33        1        2 -0.32294139
## 101       2        1  0.80152520
## 143       2        1  0.79920454
## 126       2        1  0.79869157
## 135       2        1  0.79691020
## 142       2        1  0.79676588
## 154       2        1  0.79666807
## 145       2        1  0.79639536
## 161       2        1  0.79584440
## 146       2        1  0.79473943
## 92        2        1  0.79467437
## 149       2        1  0.79455649
```

```
## 157       2        1  0.79412234
## 160       2        1  0.79410258
## 159       2        1  0.79379653
## 117       2        1  0.79332435
## 119       2        1  0.79235212
## 168       2        1  0.79181925
## 140       2        1  0.79163288
## 120       2        1  0.79158667
## 177       2        1  0.79147099
## 109       2        1  0.79106331
## 187       2        1  0.79068377
## 121       2        1  0.79068053
## 151       2        1  0.79041190
## 173       2        1  0.79011056
## 141       2        1  0.79000971
## 118       2        1  0.78982022
## 182       2        1  0.78970315
## 147       2        1  0.78968872
## 155       2        1  0.78957332
## 112       2        1  0.78954239
## 167       2        1  0.78870245
## 164       2        1  0.78839221
## 190       2        1  0.78828918
## 191       2        1  0.78782573
## 100       2        1  0.78780711
## 198       2        1  0.78699780
## 115       2        1  0.78670553
## 124       2        1  0.78653627
## 111       2        1  0.78591271
## 102       2        1  0.78559229
## 194       2        1  0.78493964
## 163       2        1  0.78467261
## 127       2        1  0.78465786
## 197       2        1  0.78450278
## 183       2        1  0.78445664
## 134       2        1  0.78345550
## 179       2        1  0.78334381
## 116       2        1  0.78238572
## 165       2        1  0.78153885
## 136       2        1  0.78089962
## 103       2        1  0.78078097
## 96        2        1  0.78076266
## 105       2        1  0.78063910
## 99        2        1  0.78049714
## 139       2        1  0.78021653
## 129       2        1  0.77936327
## 104       2        1  0.77933693
## 106       2        1  0.77923734
## 169       2        1  0.77881287
## 172       2        1  0.77847188
```

```
## 184       2         1  0.77805074
## 193       2         1  0.77804369
## 166       2         1  0.77799109
## 195       2         1  0.77788815
## 107       2         1  0.77754591
## 178       2         1  0.77745071
## 153       2         1  0.77716432
## 133       2         1  0.77365527
## 114       2         1  0.77319783
## 110       2         1  0.77187971
## 144       2         1  0.77081239
## 95        2         1  0.76606193
## 199       2         1  0.76535531
## 196       2         1  0.76471364
## 81        2         1  0.76442647
## 176       2         1  0.76353958
## 76        2         1  0.76347022
## 108       2         1  0.76062776
## 97        2         1  0.76059435
## 132       2         1  0.75996104
## 93        2         1  0.75912683
## 123       2         1  0.75848720
## 130       2         1  0.75719461
## 185       2         1  0.75641005
## 186       2         1  0.75265126
## 86        2         1  0.75261745
## 73        2         1  0.75130679
## 180       2         1  0.74994450
## 85        2         1  0.74953442
## 128       2         1  0.74856116
## 71        2         1  0.74817322
## 170       2         1  0.74797714
## 94        2         1  0.74633474
## 152       2         1  0.74274359
## 69        2         1  0.74215060
## 192       2         1  0.74111690
## 200       2         1  0.73966910
## 131       2         1  0.73939426
## 156       2         1  0.73849190
## 189       2         1  0.73724624
## 171       2         1  0.73444268
## 66        2         1  0.72914309
## 75        2         1  0.72911408
## 175       2         1  0.72663195
## 70        2         1  0.72638131
## 60        2         1  0.72608912
## 158       2         1  0.72415247
## 68        2         1  0.71992526
## 62        2         1  0.71909939
## 63        2         1  0.71799282
```

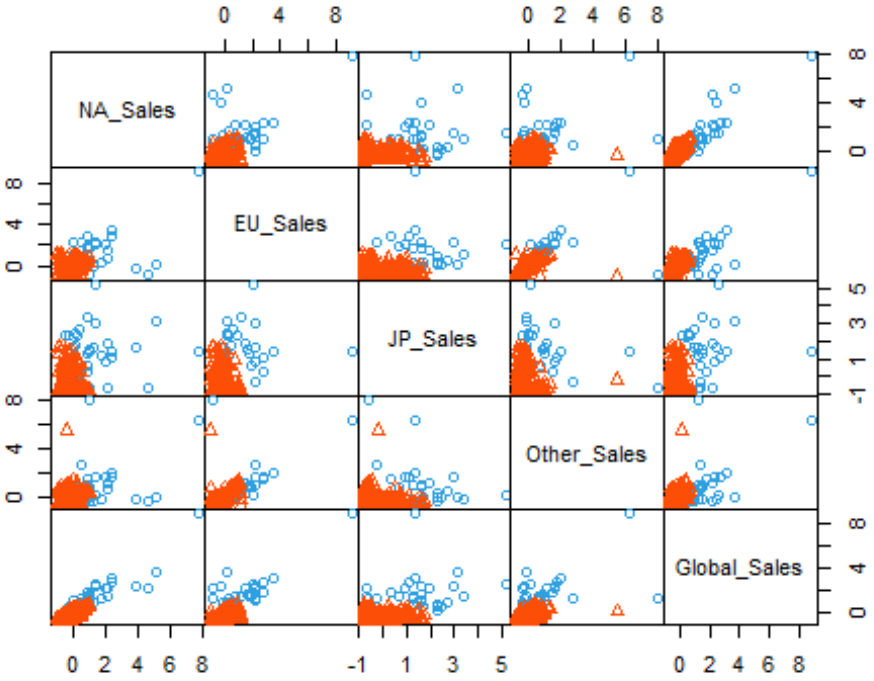
```
## 80        2        1  0.71759193
## 91        2        1  0.71689842
## 61        2        1  0.71635178
## 188       2        1  0.71378434
## 98        2        1  0.71126914
## 122       2        1  0.69774269
## 52        2        1  0.69721390
## 125       2        1  0.69411798
## 90        2        1  0.69234959
## 49        2        1  0.69123101
## 89        2        1  0.68730131
## 57        2        1  0.68097586
## 87        2        1  0.68015602
## 64        2        1  0.67831378
## 56        2        1  0.67802754
## 54        2        1  0.67781623
## 58        2        1  0.67365626
## 65        2        1  0.67284172
## 113       2        1  0.67188271
## 174       2        1  0.66980154
## 79        2        1  0.66773139
## 137       2        1  0.66390081
## 44        2        1  0.64451207
## 82        2        1  0.63784037
## 47        2        1  0.63626169
## 72        2        1  0.63270553
## 148       2        1  0.63090156
## 53        2        1  0.62370579
## 78        2        1  0.62290036
## 138       2        1  0.62004405
## 84        2        1  0.61893951
## 83        2        1  0.61706176
## 51        2        1  0.61127851
## 39        2        1  0.60474614
## 59        2        1  0.59500816
## 181       2        1  0.59295369
## 67        2        1  0.59005780
## 41        2        1  0.58855889
## 55        2        1  0.58781956
## 37        2        1  0.58705772
## 88        2        1  0.58692127
## 36        2        1  0.57925635
## 50        2        1  0.57263827
## 43        2        1  0.57169922
## 77        2        1  0.55453338
## 40        2        1  0.54673241
## 45        2        1  0.54626948
## 162       2        1  0.54231708
## 38        2        1  0.53753343
## 32        2        1  0.53595030
```

```
## 30        2        1  0.52825031
## 150       2        1  0.52019794
## 74        2        1  0.48518685
## 29        2        1  0.47845856
## 34        2        1  0.45953382
## 46        2        1  0.45900272
## 35        2        1  0.43910846
## 24        2        1  0.43554358
## 25        2        1  0.42789089
## 31        2        1  0.40369891
## 48        2        1  0.13737849
## Average silhouette width per cluster:
## [1] 0.04620125 0.71077412
## Average silhouette width of total data set:
## [1] 0.6177339
##
## Available components:
##  [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"
##  [6] "clusinfo"   "silinfo"    "diss"       "call"       "data"
```

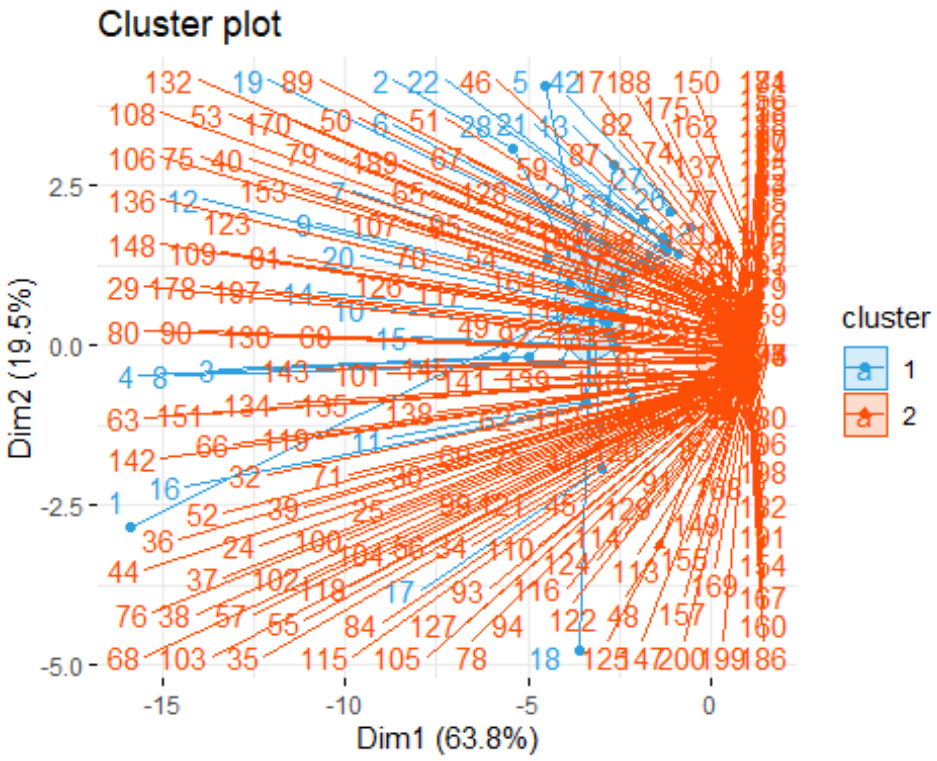
The cluster medoids us matrix which rows are the medoids and columns are variables and the clustering vector is a vector of integers indicating the cluster to which each point is allocated.

Now lets visualize the PAM clusters, first in the original space:

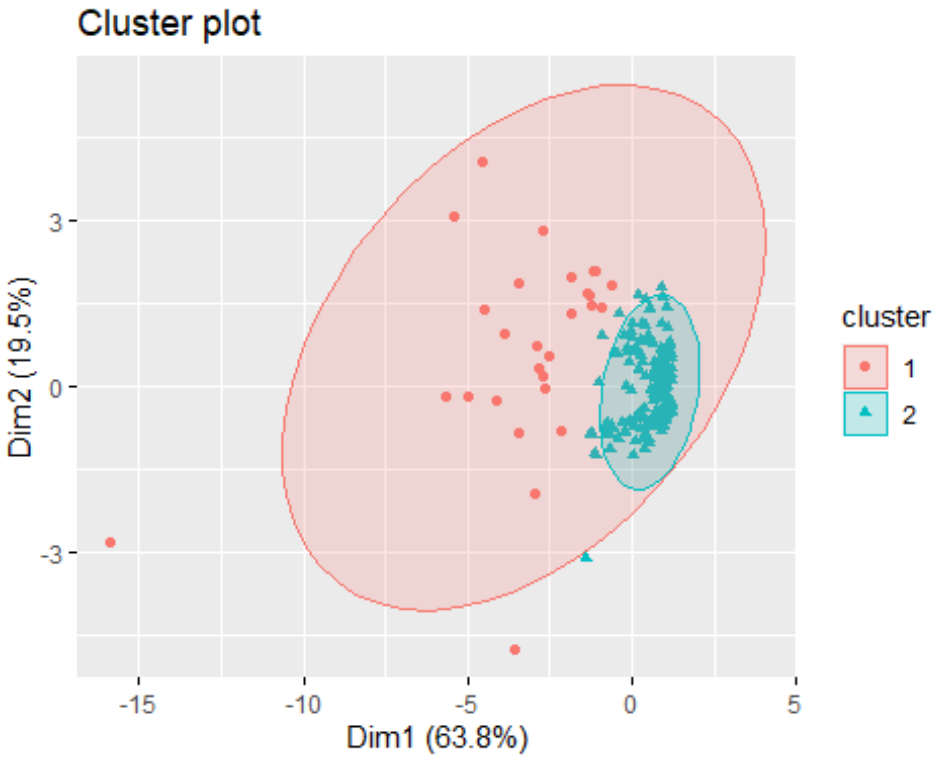
```
pam.res1 <- pam.res$cluster
pairs(vg_sales.scaled, gap=0, pch=pam.res1, col = c("#2E9FDF", "#FC4E07")
[pam.res1])
```




```
fviz_cluster(pam.res,
             data = vg_sales.scaled,
             palette = c("#2E9FDF", "#FC4E07"),
             ellipse.type = "euclid",
             star.plot = TRUE,
             repel = TRUE,
             ggtheme = theme_minimal())
```




```
fviz_cluster(pam.res, geom = "point", frame.type = "norm")
```



This method seems like the best one to this type of dataframe cause it was the only one that really avoided the issued caused by the outliers and brought us 2 good clusters that have a good partitioning. The first cluster has 172 and the second one has 28.

Cluster Validation

So far, we have applied several grouping methods, but we have not tested to see whether the dataset contains clusters or not and this may have been one of the reasons why the results were clearly not very satisfactory.

We can evaluate whether there are clusters from both a statistical and a graphical point of view, using Hopkins statistics and the VAT algorithm, respectively.

Hopkins Statistic

The Hopkins Statistic a way of measuring the cluster tendency of a data set. It acts as a statistical hypothesis test where the null hypothesis is that the data is generated by a Poisson point process and are thus uniformly randomly distributed.A value close to 1 tends to indicate the data is highly clustered, random data will tend to result in values around 0.5, and uniformly distributed data will tend to result in values close to 0.

```
library(clustertend)

hopkins(vg_sales.scaled, n = nrow(vg_sales.scaled)-1)

## $H
## [1] 0.06851391
```

The result close to 0 and it suggests that there are no clusters because the data its uniformly distributed.

Internal clustering validation measures

Internal validation measures reflect often the compactness, the connectedness and separation of the cluster partitions.

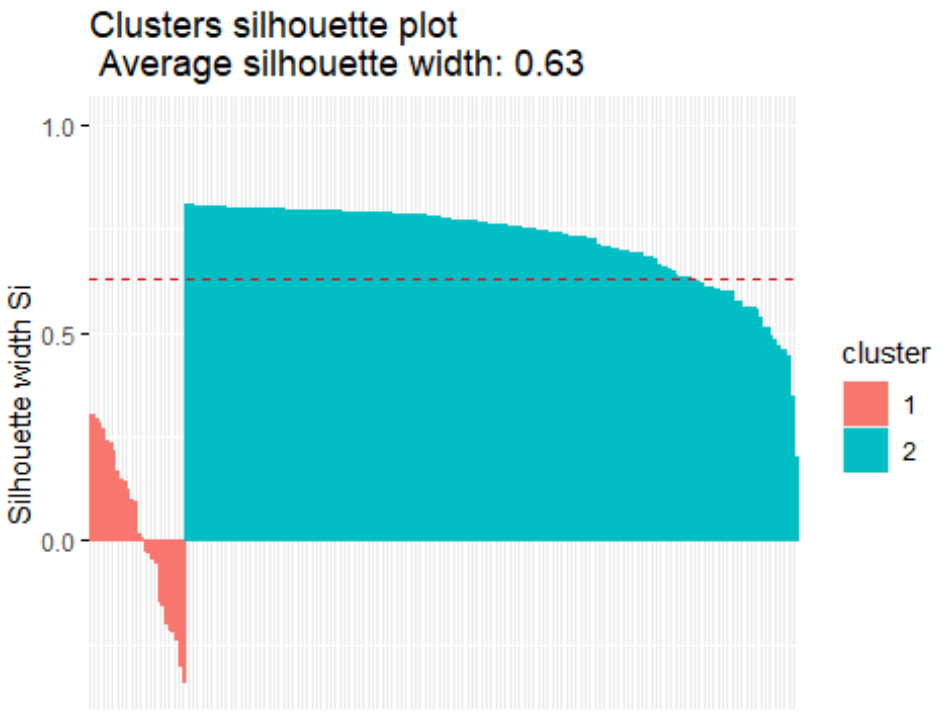
Silhouete Analysis

We described before what the silhouete analysis is to find out the good number of *k* to use in the PAM method. The Silhouette analysis measures how well an observation is clustered and it estimates the average distance between clusters. So we will calculate for each method that we used:

Silhouette analysis for average linkage method

```
s.res.alm <- eclust(vg_sales.scaled, "hclust", k = 2,
                    method = "average", graph = FALSE)
silinfo <- s.res.alm$silinfo
silinfo$avg.width
```

```
## [1] 0.6278252

fviz_silhouette(s.res.alm)

##   cluster size ave.sil.width
## 1       1   27          0.03
## 2       2  173          0.72
```




```
silinfo$clus.avg.widths

## [1] 0.02597683 0.72175533
```

It can be seen that the sample have a negative silhouette coefficient in the hierarchical clustering. This means that they are not in the right cluster.

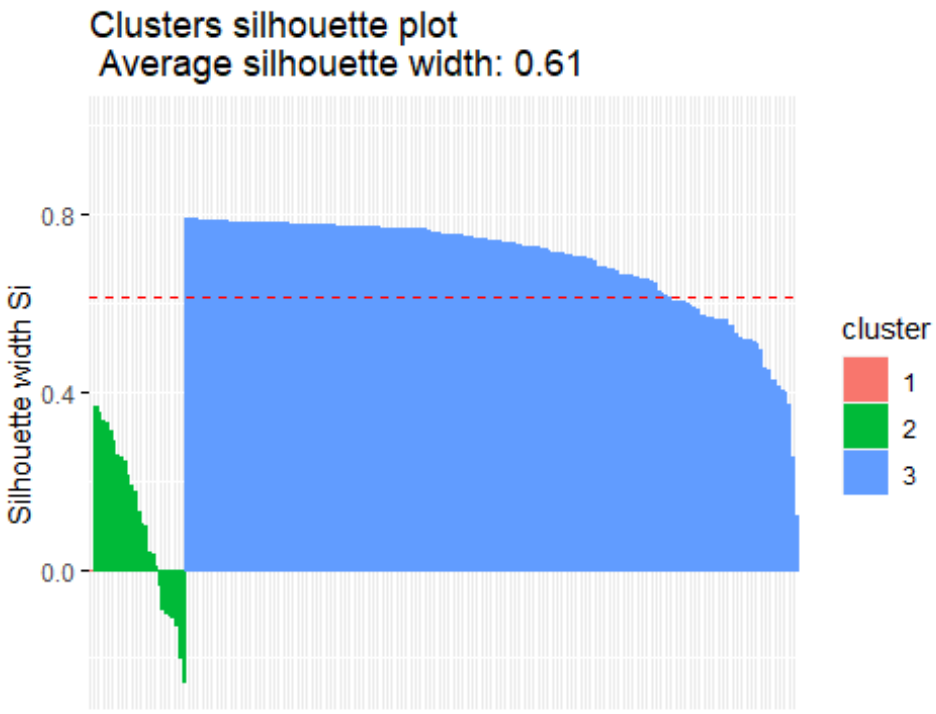
We can find the name of these samples and determine the clusters they are closer (neighbor cluster), as follow:

```
sil <- s.res.alm$silinfo$widths[, 1:3]
neg_sil_index<- which(sil[,'sil_width'] < 0)
sil[neg_sil_index, , drop = FALSE]

##    cluster neighbor   sil_width
## 15       1        2 -0.02292670
## 21       1        2 -0.02627922
## 17       1        2 -0.04187564
## 10       1        2 -0.05118642
## 48       1        2 -0.14457165
```

```
## 19       1        2 -0.15445592
## 28       1        2 -0.19926437
## 27       1        2 -0.21281081
## 26       1        2 -0.21732240
## 22       1        2 -0.23400162
## 23       1        2 -0.29909342
## 42       1        2 -0.33874168
```

Silhouette analysis for ward's linkage method

```
s.res.wlm <- eclust(vg_sales.scaled, "hclust", k = 3,
                    method = "ward", graph = FALSE)
silinfo1 <- s.res.wlm$silinfo
silinfo1$avg.width

## [1] 0.6130973

fviz_silhouette(s.res.wlm)

##   cluster size ave.sil.width
## 1       1    1          0.00
## 2       2   26          0.10
## 3       3  173          0.69
```



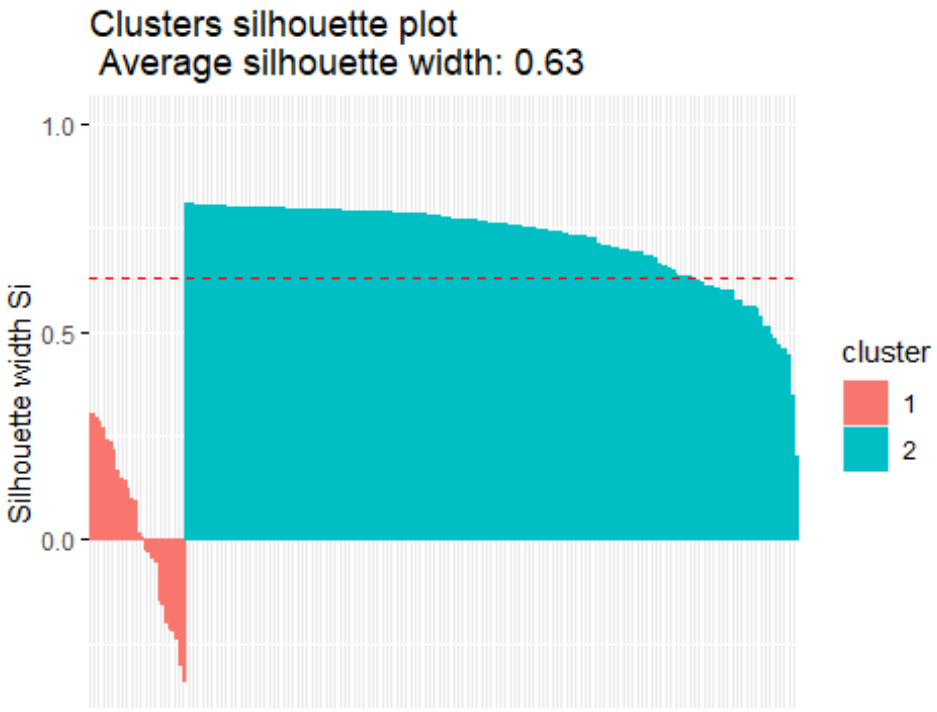

```
silinfo1$clus.avg.widths

## [1] 0.0000000 0.1047530 0.6930398
```

Now lets find the name of these samples that are not in the right cluster and determine the clusters they are closer, as follow:

```
sil1 <- s.res.wlm$silinfo$widths[, 1:3]
neg_sil_index1<- which(sil1[,'sil_width'] < 0)
sil1[neg_sil_index1, , drop = FALSE]

##    cluster neighbor   sil_width
## 19       2        3 -0.03610506
## 28       2        3 -0.08700621
## 48       2        3 -0.09774167
## 27       2        3 -0.10158901
## 26       2        3 -0.10503526
## 22       2        3 -0.12553406
## 23       2        3 -0.19706604
## 42       2        3 -0.25303010
```

Silhouette analysis for single linkage method

```
s.res.slm <- eclust(vg_sales.scaled, "hclust", k = 2,
                    method = "single", graph = FALSE)
silinfo2 <- s.res.slm$silinfo
silinfo2$avg.width

## [1] 0.6278252

fviz_silhouette(s.res.slm)

##   cluster size ave.sil.width
## 1       1   27          0.03
## 2       2  173          0.72
```




```
silinfo2$clus.avg.widths

## [1] 0.02597683 0.72175533
```

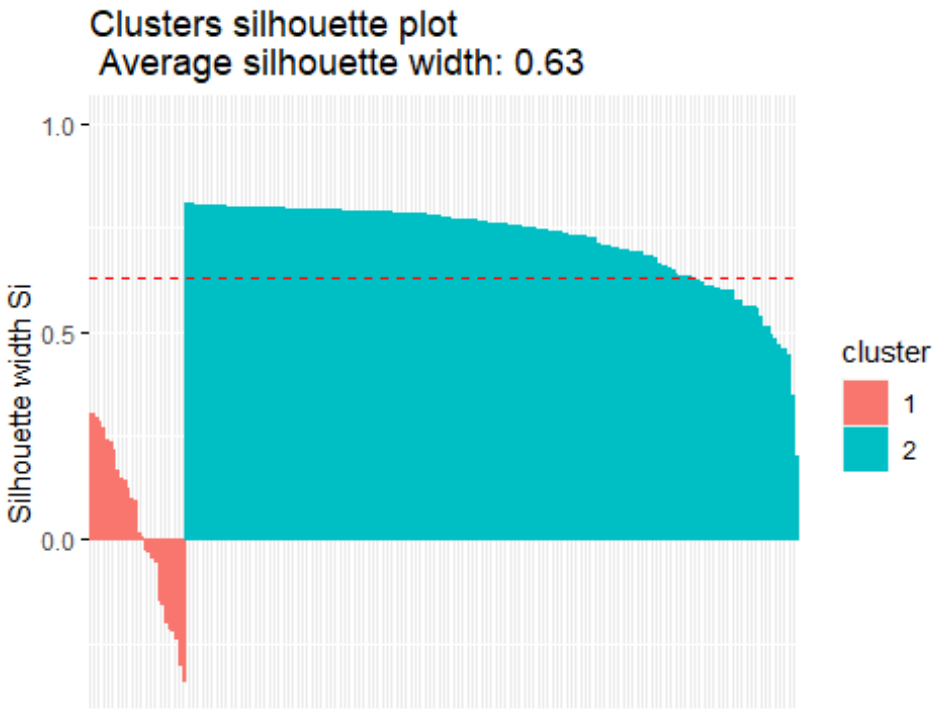
Now lets find the name of these samples that are not in the right cluster and determine the clusters they are closer, as follow:

```
sil2 <- s.res.slm$silinfo$widths[, 1:3]
neg_sil_index2 <- which(sil2['sil_width'] < 0)
sil2[neg_sil_index2, , drop = FALSE]

##    cluster neighbor   sil_width
## 15       1        2 -0.02292670
## 21       1        2 -0.02627922
## 17       1        2 -0.04187564
## 10       1        2 -0.05118642
## 48       1        2 -0.14457165
## 19       1        2 -0.15445592
## 28       1        2 -0.19926437
## 27       1        2 -0.21281081
## 26       1        2 -0.21732240
## 22       1        2 -0.23400162
## 23       1        2 -0.29909342
## 42       1        2 -0.33874168
```

Silhouette analysis for complete linkage method

```
s.res.clm <- eclust(vg_sales.scaled, "hclust", k = 2,
                    method = "complete", graph = FALSE)
```

```
silinfo3 <- s.res.clm$silinfo
silinfo3$avg.width

## [1] 0.6278252

fviz_silhouette(s.res.clm)

##   cluster size ave.sil.width
## 1       1   27          0.03
## 2       2  173          0.72
```




```
silinfo3$clus.avg.widths

## [1] 0.02597683 0.72175533
```

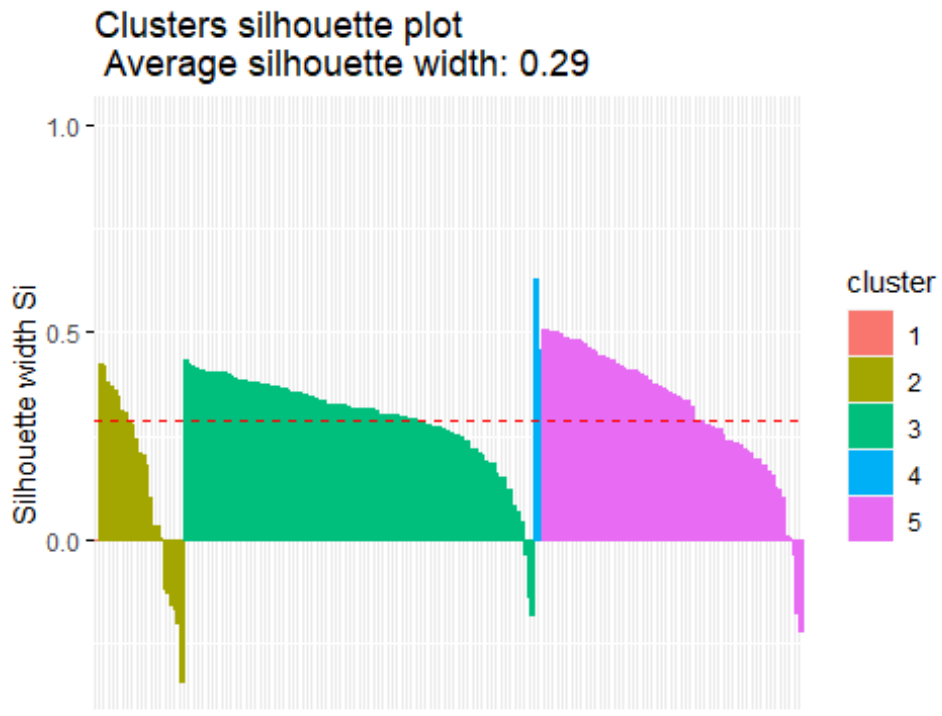
Now lets find the name of these samples that are not in the right cluster and determine the clusters they are closer, as follow:

```
sil3 <- s.res.clm$silinfo$widths[, 1:3]
neg_sil_index3<- which(sil3[,'sil_width'] < 0)
sil3[neg_sil_index3, , drop = FALSE]

##    cluster neighbor   sil_width
## 15       1        2 -0.02292670
## 21       1        2 -0.02627922
## 17       1        2 -0.04187564
## 10       1        2 -0.05118642
## 48       1        2 -0.14457165
```

```
## 19       1        2 -0.15445592
## 28       1        2 -0.19926437
## 27       1        2 -0.21281081
## 26       1        2 -0.21732240
## 22       1        2 -0.23400162
## 23       1        2 -0.29909342
## 42       1        2 -0.33874168
```

Silhouette analysis for k-means

```
s.res.km <- eclust(vg_sales.scaled, "hclust", k = 5,
                   method = "kmeans", graph = FALSE)
silinfo4 <- s.res.km$silinfo
silinfo4$avg.width

## [1] 0.2859319

fviz_silhouette(s.res.km)

##   cluster size ave.sil.width
## 1       1    1          0.00
## 2       2   24          0.14
## 3       3   99          0.29
## 4       4    2          0.54
## 5       5   74          0.32
```



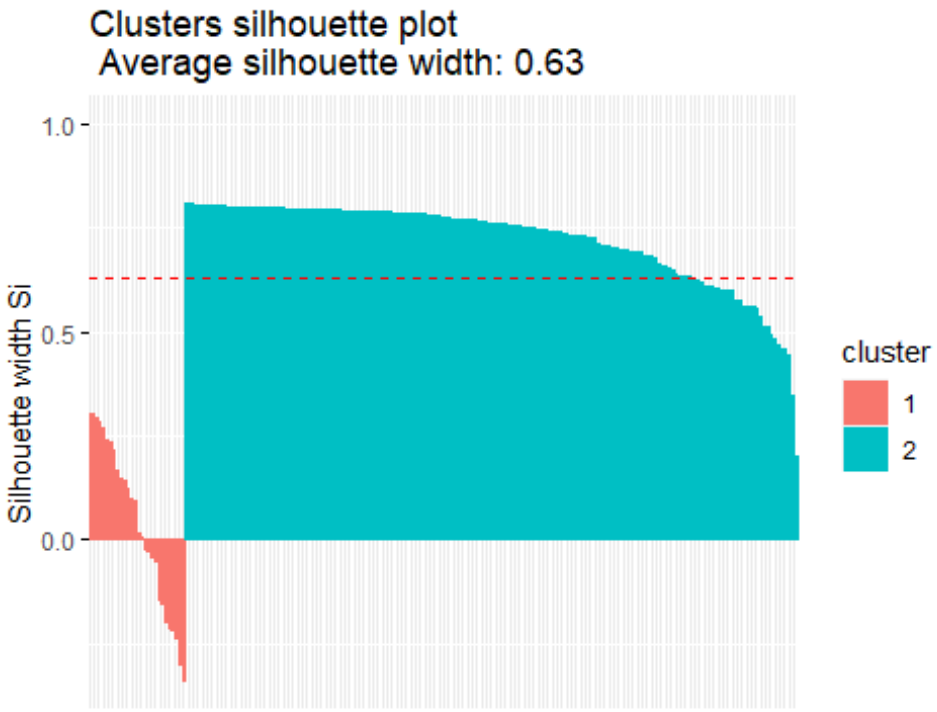

```
silinfo4$clus.avg.widths

## [1] 0.0000000 0.1404789 0.2948693 0.5416150 0.3181025
```

Now lets find the name of these samples that are not in the right cluster and determine the clusters they are closer, as follow:

```
sil4 <- s.res.km$silinfo$widths[, 1:3]
neg_sil_index4<- which(sil4[,'sil_width'] < 0)
sil4[neg_sil_index4, , drop = FALSE]

##     cluster neighbor   sil_width
## 28        2        5 -0.11842589
## 22        2        5 -0.12705852
## 26        2        5 -0.15702019
## 27        2        5 -0.16566604
## 23        2        5 -0.20293936
## 42        2        5 -0.34301059
## 97        3        5 -0.03592884
## 111       3        5 -0.13988556
## 136       3        5 -0.18152548
## 145       5        3 -0.03277388
## 190       5        3 -0.17735440
## 101       5        3 -0.21818106
```

Silhouette analysis for k-medoids (PAM)

```
s.res.pam <- eclust(vg_sales.scaled, "hclust", k = 2,
                    method = "pam", graph = FALSE)
silinfo5 <- s.res.pam$silinfo
silinfo5$avg.width

## [1] 0.6278252

fviz_silhouette(s.res.pam)

##   cluster size ave.sil.width
## 1       1   27          0.03
## 2       2  173          0.72
```




```
silinfo5 <- s.res.pam$silinfo
silinfo5$avg.width

## [1] 0.6278252
```

Now lets find the name of these samples that are not in the right cluster and determine the clusters they are closer, as follow:

```
sil5 <- s.res.pam$silinfo$widths[, 1:3]
neg_sil_index5<- which(sil5[,'sil_width'] < 0)
sil5[neg_sil_index5, , drop = FALSE]

##    cluster neighbor   sil_width
## 15       1        2 -0.02292670
## 21       1        2 -0.02627922
## 17       1        2 -0.04187564
## 10       1        2 -0.05118642
## 48       1        2 -0.14457165
## 19       1        2 -0.15445592
## 28       1        2 -0.19926437
## 27       1        2 -0.21281081
## 26       1        2 -0.21732240
## 22       1        2 -0.23400162
## 23       1        2 -0.29909342
## 42       1        2 -0.33874168
```

Almost all of the methods have the average silhouette width close to 0.6 what is closer to 1. That means that the observations are very well clustered. The Sihouette Analysis for the K-means have the average width 0.29, which is closer to 0, what means that the observations lies between two clusters.

All the methods have observations with negative width, which means that these observations are probably placed in the wrong cluster.

Dunn Index

Dunn index is a metric for evaluating clustering algorithms. Like all other indexes, the objective is to identify sets of clusters that are compact, with little variation between the members of the cluster, and well separated, where the averages of the different clusters are sufficiently distant, compared to the internal cluster. For a certain cluster assignment, a higher Dunn index indicates better grouping.

Dunn analysis for average linkage method

```
library(fpc)

di <- cluster.stats(dist(vg_sales.scaled), s.res.alm$cluster)
di$dunn

## [1] 0.04347825
```

Dunn analysis for ward's linkage method

```
di1 <- cluster.stats(dist(vg_sales.scaled), s.res.wlm$cluster)
di1$dunn

## [1] 0.06801063
```

Dunn analysis for single linkage method

```
di2 <- cluster.stats(dist(vg_sales.scaled), s.res.slm$cluster)
di2$dunn

## [1] 0.04347825
```

Dunn analysis for complete linkage method

```
di3 <- cluster.stats(dist(vg_sales.scaled), s.res.clm$cluster)
di3$dunn

## [1] 0.04347825
```

Dunn analysis for k-means

```
di4 <- cluster.stats(dist(vg_sales.scaled), s.res.km$cluster)
di4$dunn

## [1] 0.0289437
```

Dunn analysis for k-medoids (PAM)

```
di5 <- cluster.stats(dist(vg_sales.scaled), s.res.pam$cluster)
di5$dunn

## [1] 0.04347825
```

According to the Dunn index, all the units are not clustered well enough because all the index are very low.

External Validation measures

The aim of the external validation measures is to compare the identified clusters to a reference. In this case we will use the categorical variable from our dataset *Genre*.

External analysis for average linkage method

```
table(vg_sales$Genre, s.res.alm$cluster)

##
##                 1  2
##   Action        2 34
##   Adventure     0  2
##   Fighting      0  7
##   Misc          2 13
##   Platform      6 25
##   Puzzle        2  4
##   Racing        3 13
##   Role-Playing  5 25
##   Shooter       1 34
##   Simulation    2  5
##   Sports        4 10
##   Strategy      0  1
```

According to the Confusion matrix, the number of clusters is not equal to the to categorical values. The number of clusters is 2 and the categorical variables are 12.

Now lets check the correct rand index:

```
genre <- as.numeric(vg_sales$Genre)
extern <- cluster.stats(d = dist(vg_sales), genre, s.res.alm$cluster)

extern$corrected.rand

## [1] 0.01245392
```

External analysis for ward's linkage method

```
table(vg_sales$Genre, s.res.wlm$cluster)

##
##                 1  2  3
##   Action        0  2 34
```

```
##   Adventure     0  0  2
##   Fighting      0  0  7
##   Misc          0  2 13
##   Platform      0  6 25
##   Puzzle        0  2  4
##   Racing        0  3 13
##   Role-Playing  0  5 25
##   Shooter       0  1 34
##   Simulation    0  2  5
##   Sports        1  3 10
##   Strategy      0  0  1
```

According to the Confusion matrix, the number of clusters is not equal to the to categorical values. The number of clusters is 3 and the categorical variables are 12.

Now lets check the correct rand index:

```
extern1 <- cluster.stats(d = dist(vg_sales), genre, s.res.wlm$cluster)

extern1$corrected.rand

## [1] 0.01250956
```

External analysis for single linkage method

```
table(vg_sales$Genre, s.res.slm$cluster)

##
##                 1  2
##   Action        2 34
##   Adventure     0  2
##   Fighting      0  7
##   Misc          2 13
##   Platform      6 25
##   Puzzle        2  4
##   Racing        3 13
##   Role-Playing  5 25
##   Shooter       1 34
##   Simulation    2  5
##   Sports        4 10
##   Strategy      0  1
```

According to the Confusion matrix, the number of clusters is not equal to the to categorical values. The number of clusters is 2 and the categorical variables are 12.

Now lets check the correct rand index:

```
extern2 <- cluster.stats(d = dist(vg_sales), genre, s.res.slm$cluster)

extern2$corrected.rand

## [1] 0.01245392
```

External analysis for complete linkage method

```
table(vg_sales$Genre, s.res.clm$cluster)

##
##                 1  2
##   Action        2 34
##   Adventure     0  2
##   Fighting      0  7
##   Misc          2 13
##   Platform      6 25
##   Puzzle        2  4
##   Racing        3 13
##   Role-Playing  5 25
##   Shooter       1 34
##   Simulation    2  5
##   Sports        4 10
##   Strategy      0  1
```

According to the Confusion matrix, the number of clusters is not equal to the to categorical values. The number of clusters is 2 and the categorical variables are 12.

Now lets check the correct rand index:

```
extern3 <- cluster.stats(d = dist(vg_sales), genre, s.res.clm$cluster)

extern3$corrected.rand

## [1] 0.01245392
```

External analysis for k-means

```
table(vg_sales$Genre, s.res.km$cluster)

##
##                 1  2  3  4  5
##   Action        0  1 24  1 10
##   Adventure     0  0  1  0  1
##   Fighting      0  0  0  0  7
##   Misc          0  2  8  0  5
##   Platform      0  6  7  0 18
##   Puzzle        0  2  1  0  3
##   Racing        0  2  6  1  7
##   Role-Playing  0  5  6  0 19
##   Shooter       0  1 34  0  0
##   Simulation    0  2  3  0  2
##   Sports        1  3  9  0  1
##   Strategy      0  0  0  0  1
```

According to the Confusion matrix, the number of clusters is not equal to the to categorical values. The number of clusters is 5 and the categorical variables are 12.

Now lets check the correct rand index:

```
extern4 <- cluster.stats(d = dist(vg_sales), genre, s.res.km$cluster)

extern4$corrected.rand

## [1] 0.102575
```

External analysis for k-medoids (PAM)

```
table(vg_sales$Genre, s.res.pam$cluster)

##
##                 1  2
##   Action        2 34
##   Adventure     0  2
##   Fighting      0  7
##   Misc          2 13
##   Platform      6 25
##   Puzzle        2  4
##   Racing        3 13
##   Role-Playing  5 25
##   Shooter       1 34
##   Simulation    2  5
##   Sports        4 10
##   Strategy      0  1
```

According to the Confusion matrix, the number of clusters is not equal to the to categorical values. The number of clusters is 2 and the categorical variables are 12.

Now lets check the correct rand index:

```
extern5 <- cluster.stats(d = dist(vg_sales), genre, s.res.pam$cluster)

extern5$corrected.rand

## [1] 0.01245392
```

The corrected Rand index provides a measure for assessing the similarity between two partitions, adjusted for chance. Its range is -1 (no agreement) to 1 (perfect agreement). Agreement between the genres and the cluster solution are very close to zero which means that there is no agreement between the numerical value and the cluster solution method.

Choosing the best Clustering Algorithm

To choose the best pair of clustering algorithm and optimal number of clusters, we will use internal measures and stability ones.

Internal Measures

```
library(clValid)
```

```
clmethods <- c ("hierarchical", "kmeans", "pam")
intern_vg.eucl<-clValid(vg_sales.scaled, nClust=2:5, clMethods= clmethods
, metric="euclidean", validation="internal")

summary(intern_vg.eucl)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5
##
## Validation Measures:
##                                  2       3       4       5
##
## hierarchical Connectivity   2.9290  6.7869 14.5389 14.7056
##              Dunn           1.0245  0.5411  0.3153  0.3283
##              Silhouette     0.8604  0.7181  0.6713  0.6526
## kmeans       Connectivity   2.9290  6.7869 23.5226 33.4611
##              Dunn           1.0245  0.5411  0.0832  0.0737
##              Silhouette     0.8604  0.7181  0.6116  0.5257
## pam          Connectivity  16.9381 42.0290 57.9889 84.7298
##              Dunn           0.0375  0.0106  0.0048  0.0046
##              Silhouette     0.6177  0.3215  0.2082  0.2135
##
## Optimal Scores:
##
##              Score  Method       Clusters
## Connectivity 2.9290 hierarchical 2
## Dunn         1.0245 hierarchical 2
## Silhouette   0.8604 hierarchical 2
```

According to this method, the best clustering method using the euclidean distance is:

> According to the connectivity, that must be minimized, the best solution would be the hierchical.

> According to the Dunn value that must be maximized, the best solution is the hierachical.

> According the silhouette, that also must be maximized, the best solution is the hierachical as well.

So, we can say that the best clustering method based on the internal measures and using euclidean distance is the hierarchical with 2 clusters.

Now lets try using the manhattan distance:

```
clmethods1 <- c ("hierarchical", "kmeans", "pam")
intern_vg.man<-clValid(vg_sales.scaled, nClust=2:5, clMethods= clmethods,
metric="manhattan", validation="internal")
```

```
summary(intern_vg.eucl)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5
##
## Validation Measures:
##                                  2       3       4       5
##
## hierarchical Connectivity   2.9290  6.7869 14.5389 14.7056
##              Dunn           1.0245  0.5411  0.3153  0.3283
##              Silhouette     0.8604  0.7181  0.6713  0.6526
## kmeans       Connectivity   2.9290  6.7869 23.5226 33.4611
##              Dunn           1.0245  0.5411  0.0832  0.0737
##              Silhouette     0.8604  0.7181  0.6116  0.5257
## pam          Connectivity  16.9381 42.0290 57.9889 84.7298
##              Dunn           0.0375  0.0106  0.0048  0.0046
##              Silhouette     0.6177  0.3215  0.2082  0.2135
##
## Optimal Scores:
##
##              Score  Method       Clusters
## Connectivity 2.9290 hierarchical 2
## Dunn         1.0245 hierarchical 2
## Silhouette   0.8604 hierarchical 2
```

According to this method, the best clustering method using the manhattan distance is:

- According to the connectivity, that must be minimized, the best solution would be the hierchical.
- According to the Dunn value that must be maximized, the best solution is the hierachical.
- According the silhouette, that also must be maximized, the best solution is the hierachical as well.

So, we can say that the best clustering method based on the internal measures and using manhattan distance is the hierarchical with 2 clusters, exactly the same thing as using the euclidean distance

Stability Measures

```
clmethods2 <- c("hierarchical","kmeans","pam")

stab <- clValid(vg_sales.scaled, nClust = 2:5, clMethods = clmethods, val
idation = "stability")
```

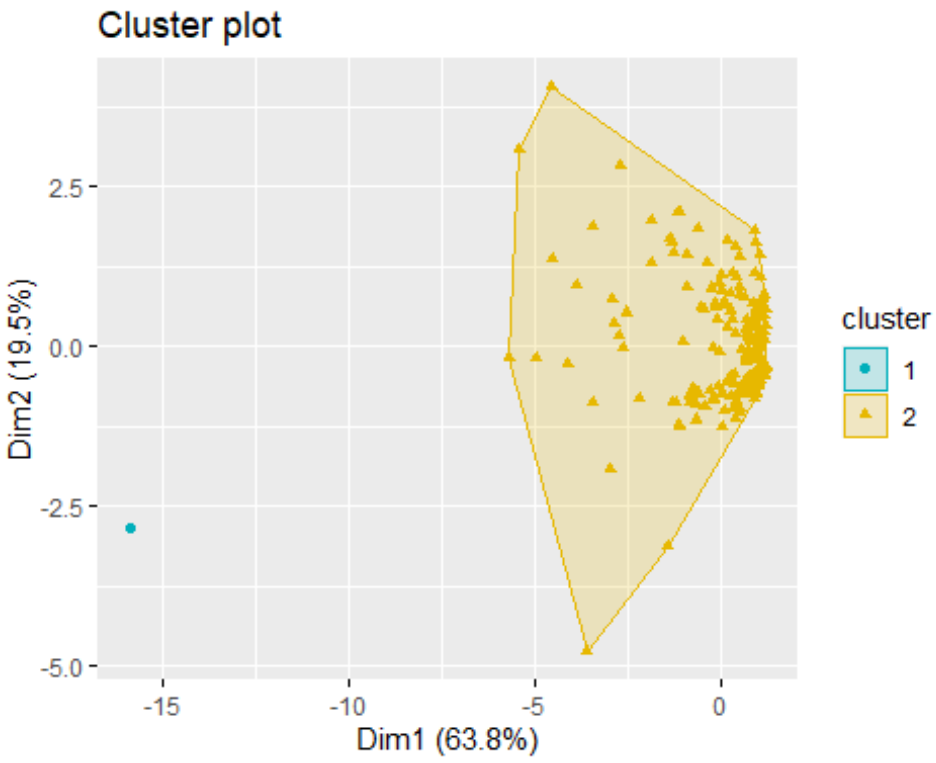
```
optimalScores(stab)

##         Score       Method Clusters
## APN 0.0000000 hierarchical        2
## AD  1.5168580          pam        5
## ADM 0.0000000 hierarchical        2
## FOM 0.6939423       kmeans        5
```

With this measure, according with APN and ADM, the best clustering method is the hierarchical with 2 clusters, according the ADM, the best clustering method is PAM with 5 clusters and according to FOM the best clustering method is kmeans with 5 clusters.

The Best Clustering Method

According to these approaches, it becomes clear that the best combination of method and number of clusters is the hierarchical one with 2 clusters. I could choose any linkage method in this case (except the wards one) because it all lead the same clustering result.

```
fviz_cluster(list(data=vg_sales.scaled, cluster =  d.alm),geom="point",el
lipse.type="convex",palette = c("#00AFBB", "#E7B800"),repel = TRUE, show.
clust.cent = FALSE)
```



But I still believe that the PAM method was the only one that really avoided the issue caused by the outliers and brought an acceptable partitioning using the optimal number of clusters that is 2 as we can see in the graph below:

```
fviz_cluster(pam.res, geom = "point", frame.type = "norm")
```