homework5

2022-11-15

1) Find a good clustering, i.e., one of which you think that it captures in the best possible manner a really meaningful grouping. Try out at least two clusterings, of which at least one is based on a Gaussian mixture, and at least one is from a different approach.

```
glass <- read.csv("C:/Users/Utente/OneDrive/Desktop/bigData/glass.dat", sep="")
data <- as.matrix(glass)

library(fpc)
library(smacof)
```

```
## Caricamento del pacchetto richiesto: plotrix

## Caricamento del pacchetto richiesto: colorspace

## Caricamento del pacchetto richiesto: e1071

##
## Caricamento pacchetto: 'smacof'

## Il seguente oggetto è mascherato da 'package:base':
##
##     transform
```

```
library(cluster)
library(pdfCluster)
```

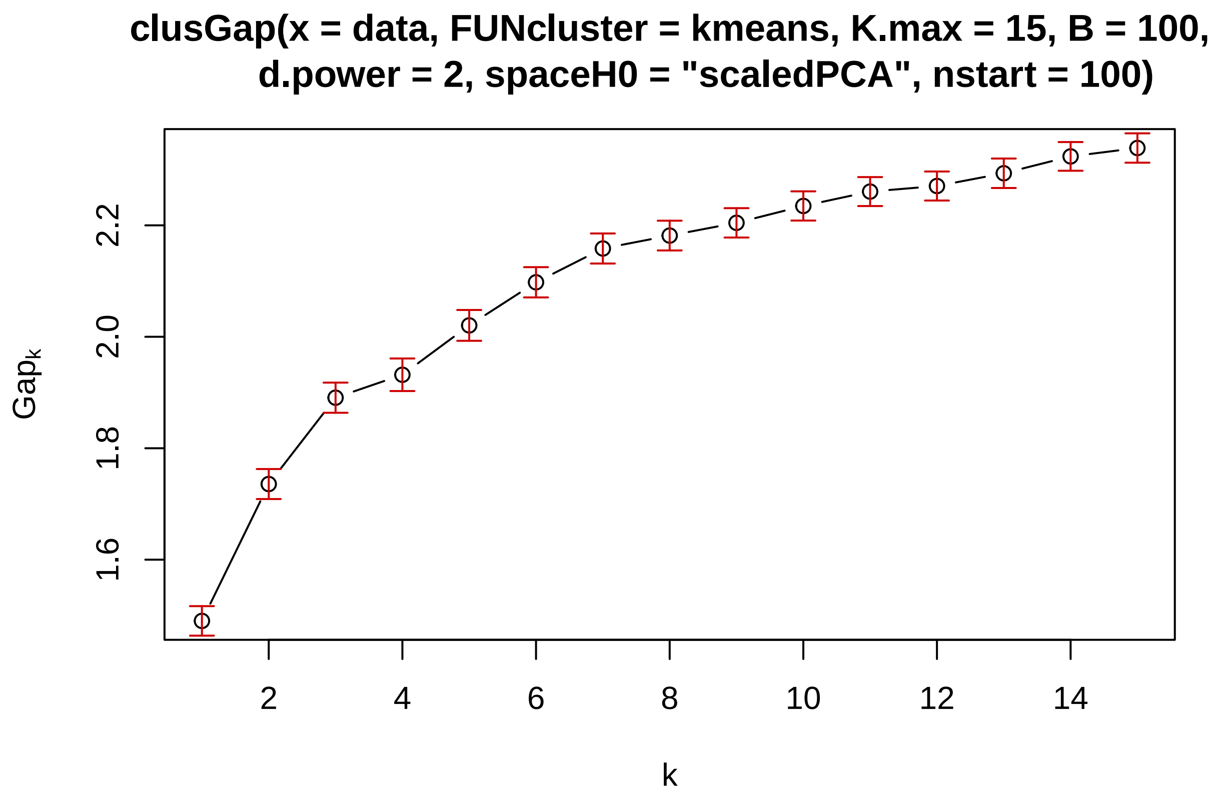
```
## pdfCluster 1.0-3
```

Clustering methods with k-means.

```
set.seed(1234)
glass.k <- clusGap(data,kmeans,K.max=15,B=100,d.power=2,spaceH0="scaledPCA",nstart=100)
print(glass.k,method="globalSEmax",SE.factor=2)
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = data, FUNcluster = kmeans, K.max = 15, B = 100, d.power = 2, spaceH0 = "scaledPCA", nsta
## B=100 simulated reference sets, k = 1..15; spaceH0="scaledPCA"
##  --> Number of clusters (method 'globalSEmax', SE.factor=2): 13
##           logW   E.logW      gap     SE.sim
##  [1,] 6.509333 7.999504 1.490171 0.02649254
##  [2,] 6.015705 7.751458 1.735753 0.02690103
##  [3,] 5.685332 7.576036 1.890704 0.02707077
##  [4,] 5.499298 7.431142 1.931844 0.02923079
##  [5,] 5.298965 7.319417 2.020452 0.02766054
##  [6,] 5.124144 7.221937 2.097793 0.02714393
##  [7,] 4.984477 7.142897 2.158420 0.02698356
##  [8,] 4.893086 7.074694 2.181608 0.02669263
##  [9,] 4.809542 7.014019 2.204476 0.02633996
## [10,] 4.723794 6.958558 2.234764 0.02623507
## [11,] 4.647796 6.908356 2.260560 0.02598480
## [12,] 4.591305 6.861847 2.270542 0.02606924
## [13,] 4.525013 6.818432 2.293418 0.02644271
## [14,] 4.453812 6.777491 2.323679 0.02573578
## [15,] 4.400417 6.739153 2.338736 0.02634792
```

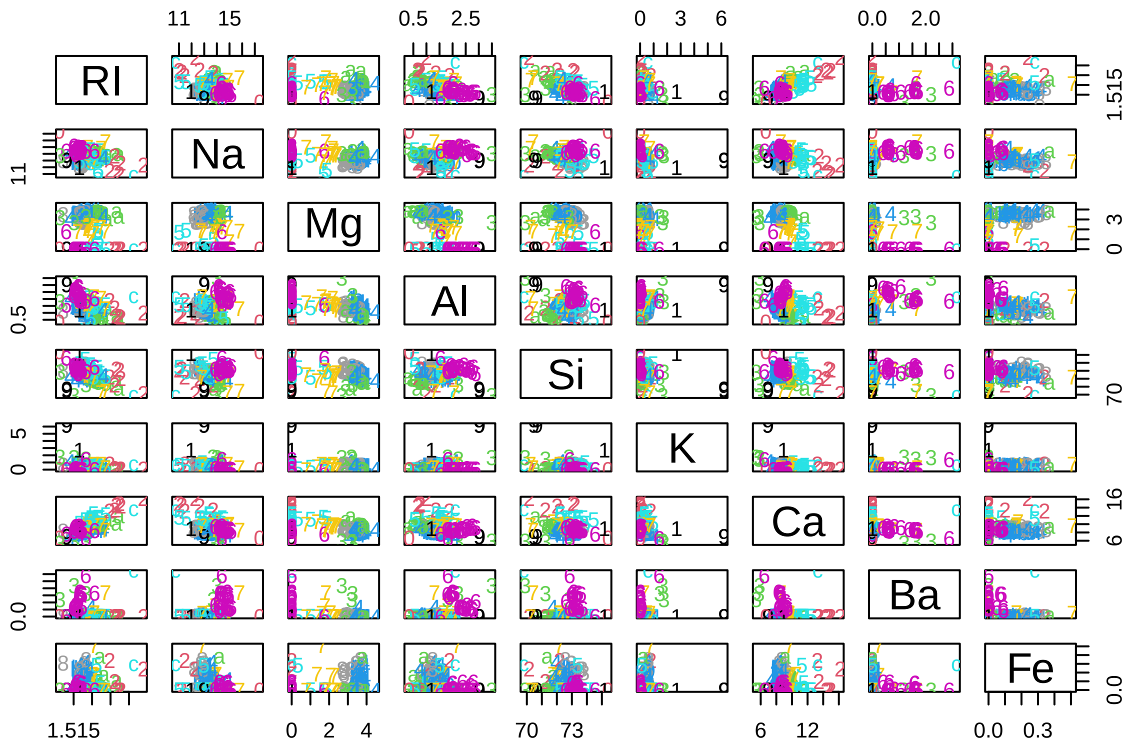
```
plot(glass.k)
```



```
cluster.k<-kmeans(data, centers = 13, nstart = 100)
str(cluster.k)
```

```
## List of 9
##  $ cluster     : int [1:214] 4 4 12 4 12 12 12 12 4 12 ...
##  $ centers     : num [1:13, 1:9] 1.52 1.53 1.51 1.52 1.52 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:13] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:9] "RI" "Na" "Mg" "Al" ...
## $ totss       : num 1343
## $ withinss    : num [1:13] 0 19.4 6.9 27.7 32 ...
## $ tot.withinss: num 184
## $ betweenss   : num 1159
## $ size        : int [1:13] 1 6 3 51 13 23 13 42 2 1 ...
## $ iter        : int 3
## $ ifault      : int 0
## - attr(*, "class")= chr "kmeans"
```

```
pairs(data,col=cluster.k$cluster,pch=clusym[cluster.k$cluster])
```



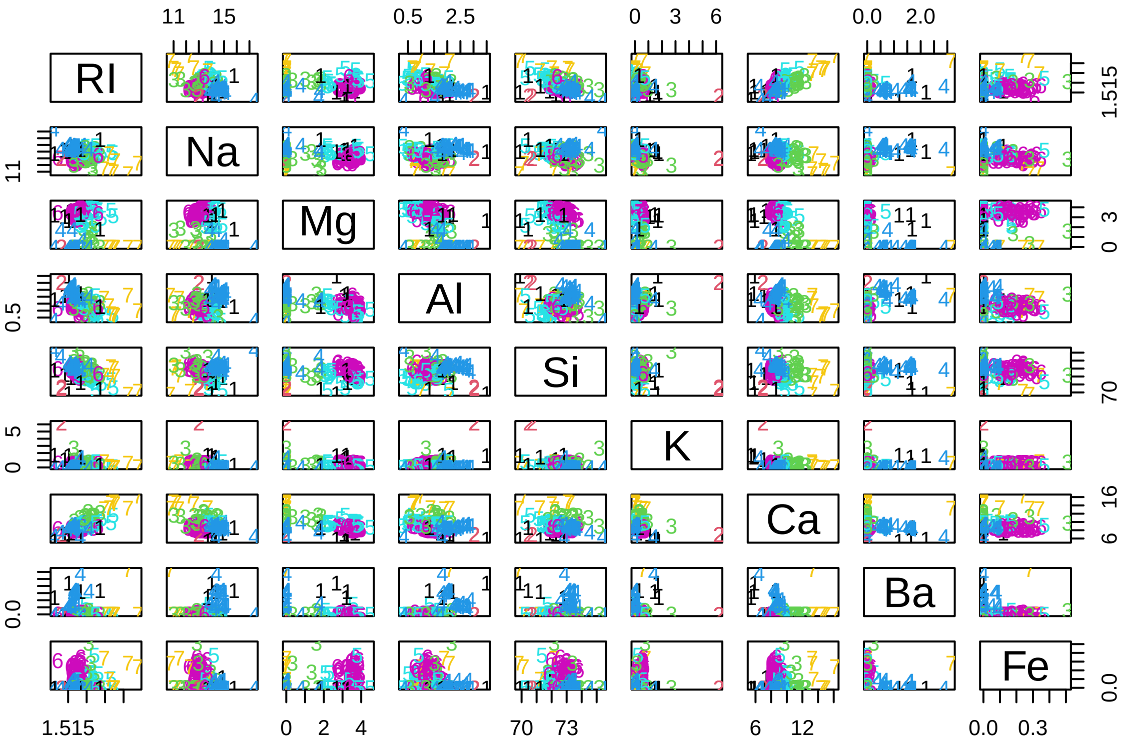
According to the gap results the best number of clusters is 13. .The refractive index is almost the same in all clusters. .The highest concentration of Sodium can be seen in the cluster 4 while the lowest is in the number 12. .The magnesium is totally absent in the clusters number 4,8,10,12. .Alluminium takes similar values in all clusters. .The silicon in high very high and takes the maximum value in the fourth cluster and the minimum in the 12. .The potassio views a very high value with respect to the others in the eigth cluster and it is very low or absent in the ninth and fourth. .calcium is high in the cluster 10 and low in the 2,4,8. .the barium is totally absent in clusters 4,7,8,10,11,13 and it takes higher value with respect to the others in the cluster 12. .The Iron is absent in clusters 2,4,8,13. We can say that the cluster number 12 take higher values for the presence of a lot of elements while the 4 is characterized for the big majority by Sodium, Silicon, and calcium.

Another well value to take as number of cluster is 7 and we can see it in the plot.

```
cluster.k7<-kmeans(data, centers = 7, nstart = 100)
str(cluster.k7)
```

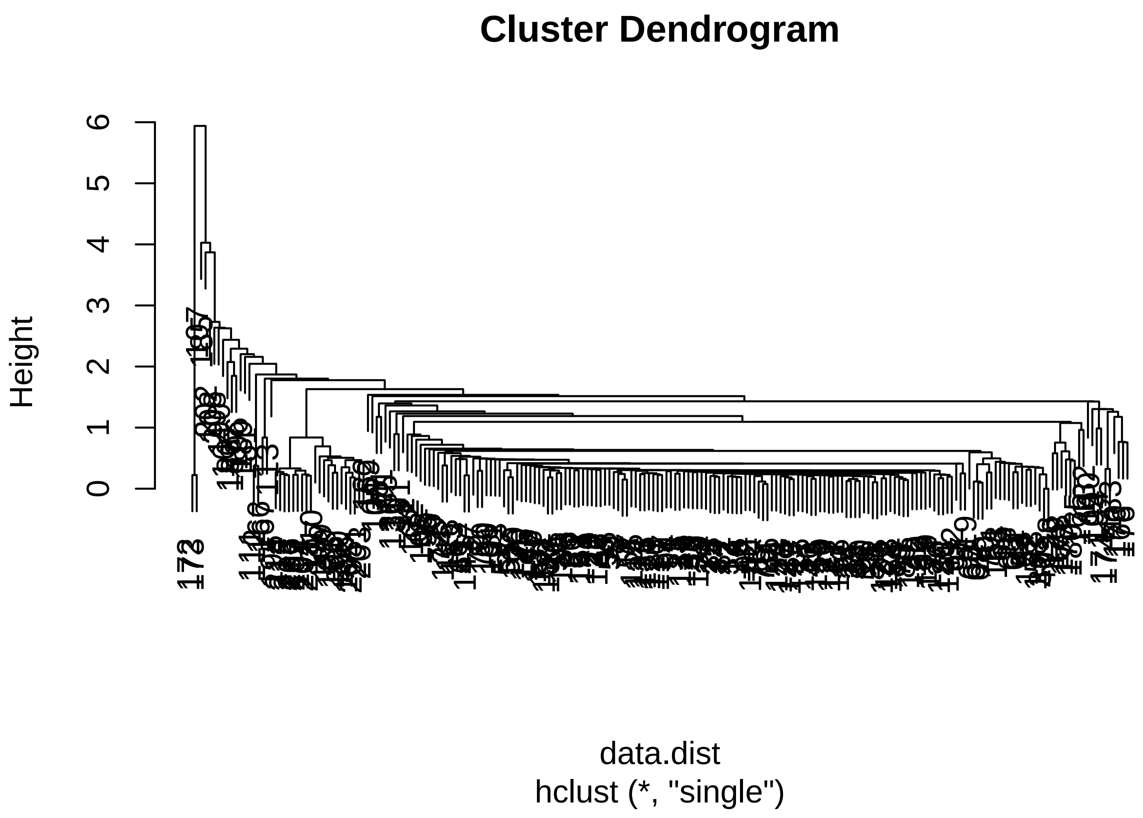
```
## List of 9
##  $ cluster     : int [1:214] 5 6 6 6 6 6 6 6 6 6 ...
##  $ centers     : num [1:7, 1:9] 1.52 1.51 1.52 1.52 1.52 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:7] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:9] "RI" "Na" "Mg" "Al" ...
##  $ totss       : num 1343
##  $ withinss    : num [1:7] 27.8733 0.0251 60.9228 59.5983 48.3074 ...
##  $ tot.withinss: num 292
##  $ betweenss   : num 1051
##  $ size        : int [1:7] 6 2 17 26 35 121 7
##  $ iter        : int 4
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

```
pairs(data,col=cluster.k7$cluster,pch=clusym[cluster.k7$cluster])
```

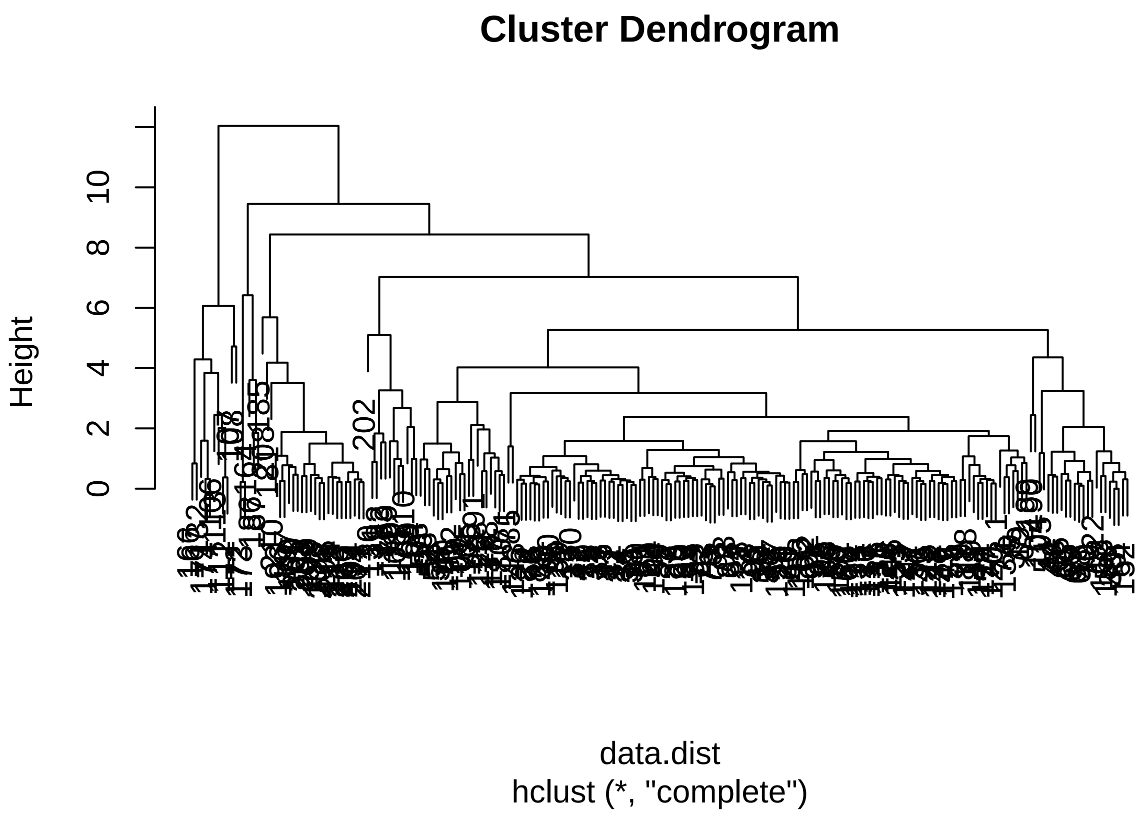


Hierarchical method.

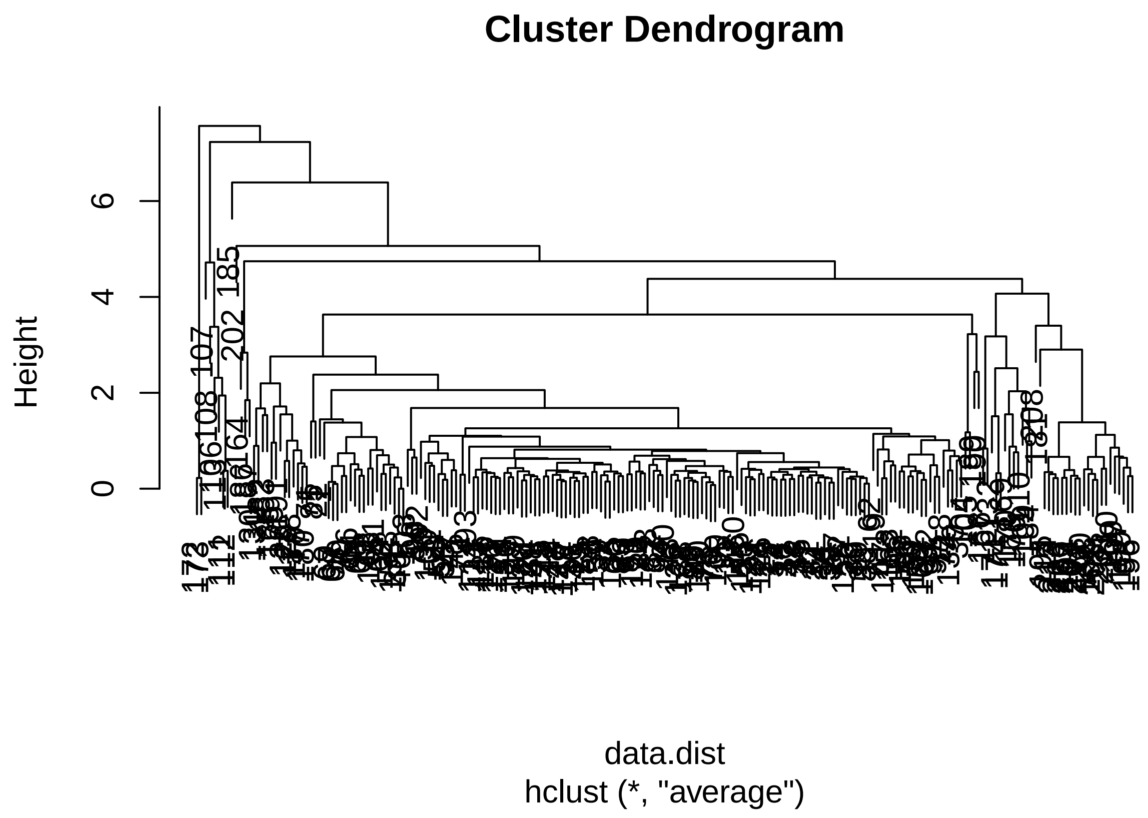
```
data.dist<-dist(data, method = "euclidean")

single<-hclust(data.dist,method = "single")
plot(single)
```



```
complete<-hclust(data.dist,method = "complete")
plot(complete)
```

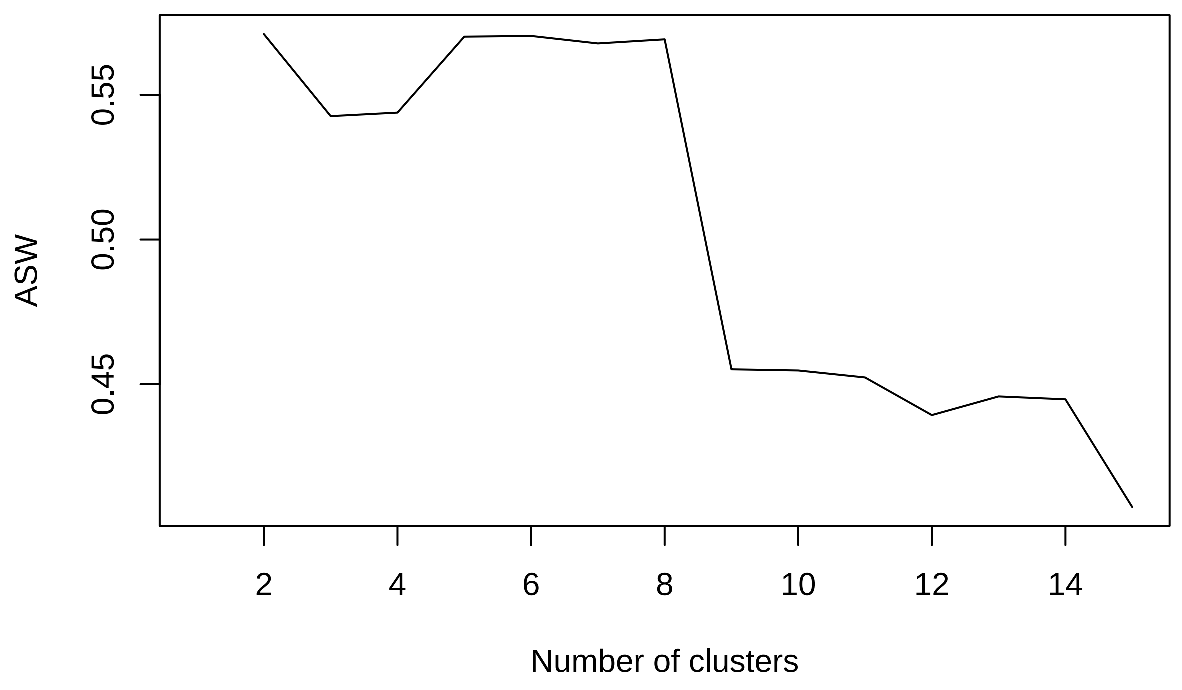


```r
average<-hclust(data.dist,method = "average")
plot(average)
```



The complete method seems to be the best because it does not create too much clusters with a single unit and it can end up with homogeneous cluster within and heterogeneous between.

```
pasw<- NA
pclusk <- list()
psil<- list()
for (k in 2:15){
  pclusk[[k]] <- cutree(complete,k)
  psil[[k]] <- silhouette(pclusk[[k]],dist=data.dist)
  pasw[k] <- summary(psil[[k]])$avg.width
}
plot(1:15,pasw,type="l",xlab="Number of clusters",ylab="ASW")
```



```
pasw
```

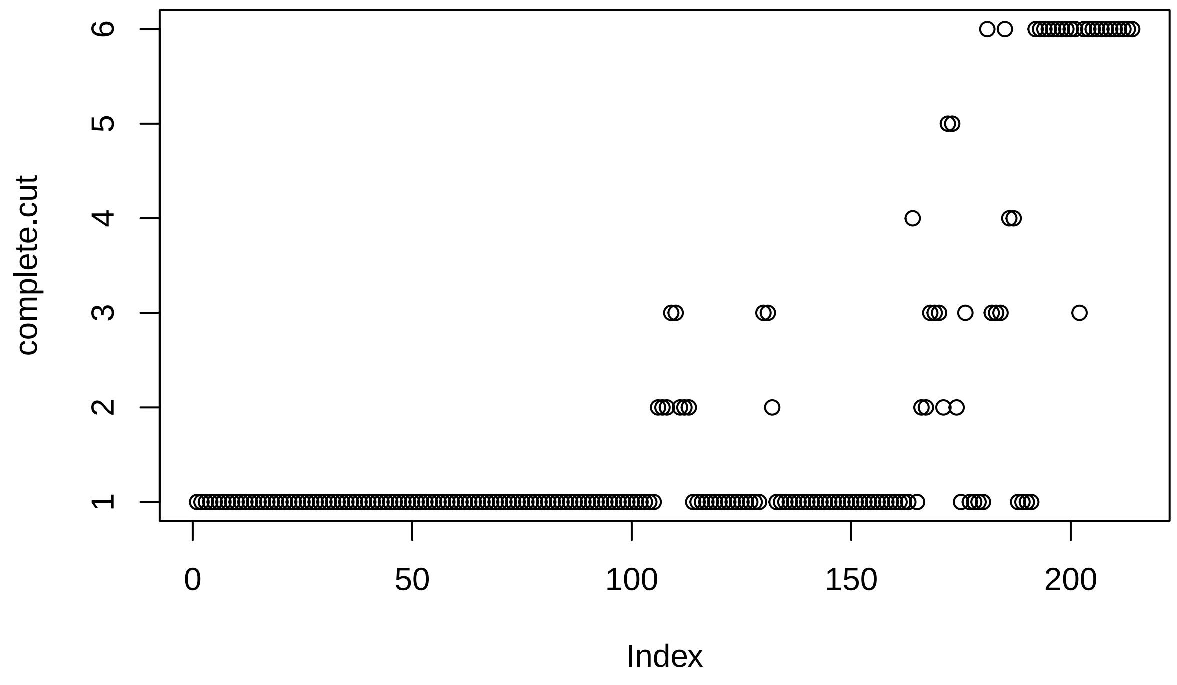
```
##  [1]        NA 0.5709892 0.5426460 0.5438598 0.5701057 0.5703585 0.5677699
##  [8] 0.5691895 0.4551557 0.4547512 0.4523441 0.4393405 0.4457849 0.4447801
## [15] 0.4075765
```

```
which.max(pasw)
```

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

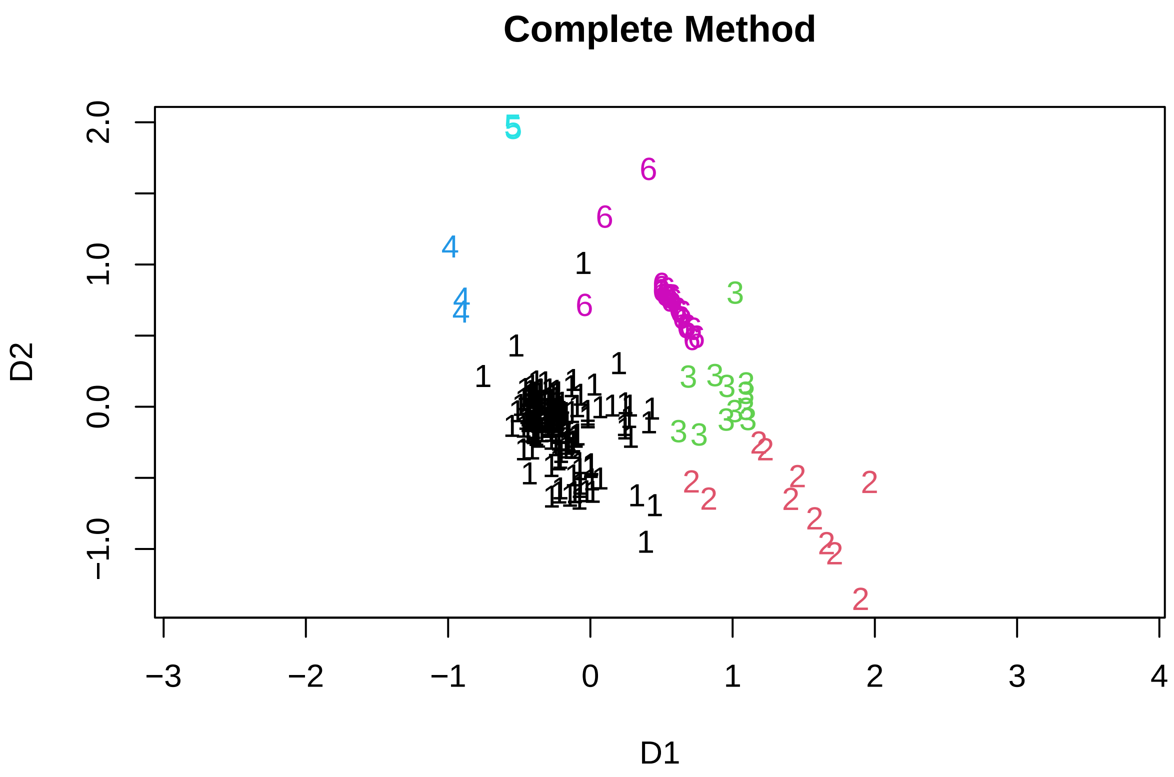
The highest value for the silhouette index corresponds to k=2 but k=6 is high and it can be considered good. [1] NA 0.5709892 0.5426460 0.5438598 0.5701057 0.5703585 0.5677699

```
complete.cut<-cutree(complete,6)
plot(complete.cut)
```

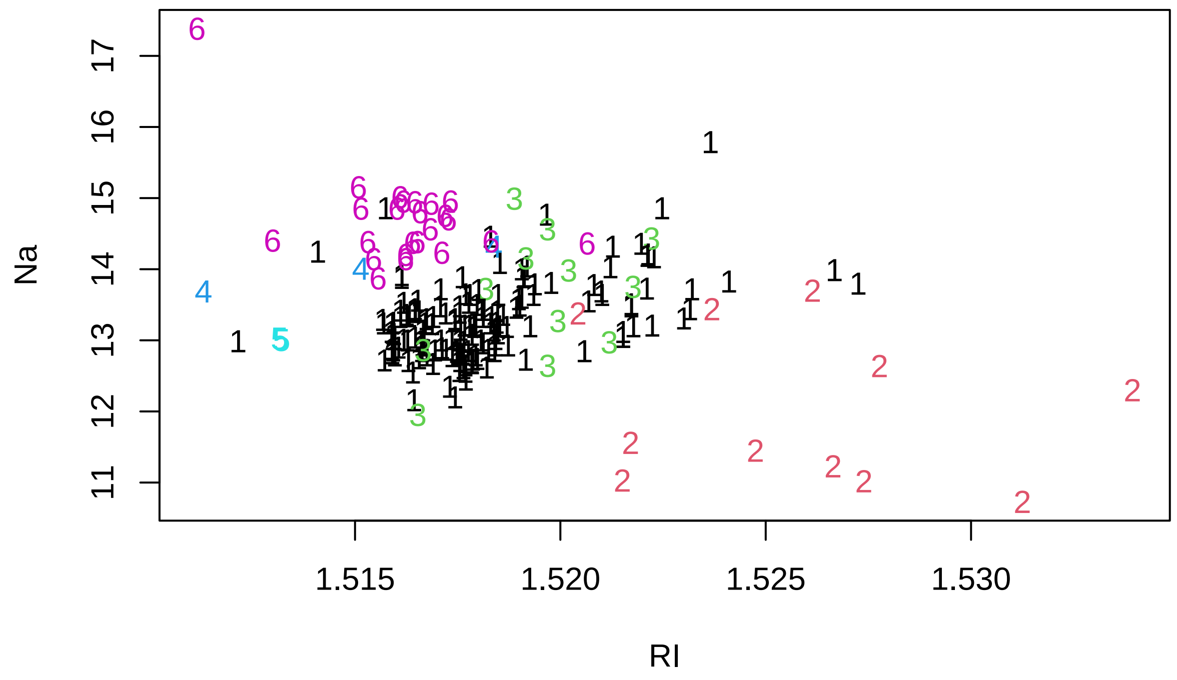


```r
mds.complete<-mds(data.dist)
plot(mds.complete$conf, col=complete.cut, pch=clusym[complete.cut],asp=1, main= "Complete Method")
```

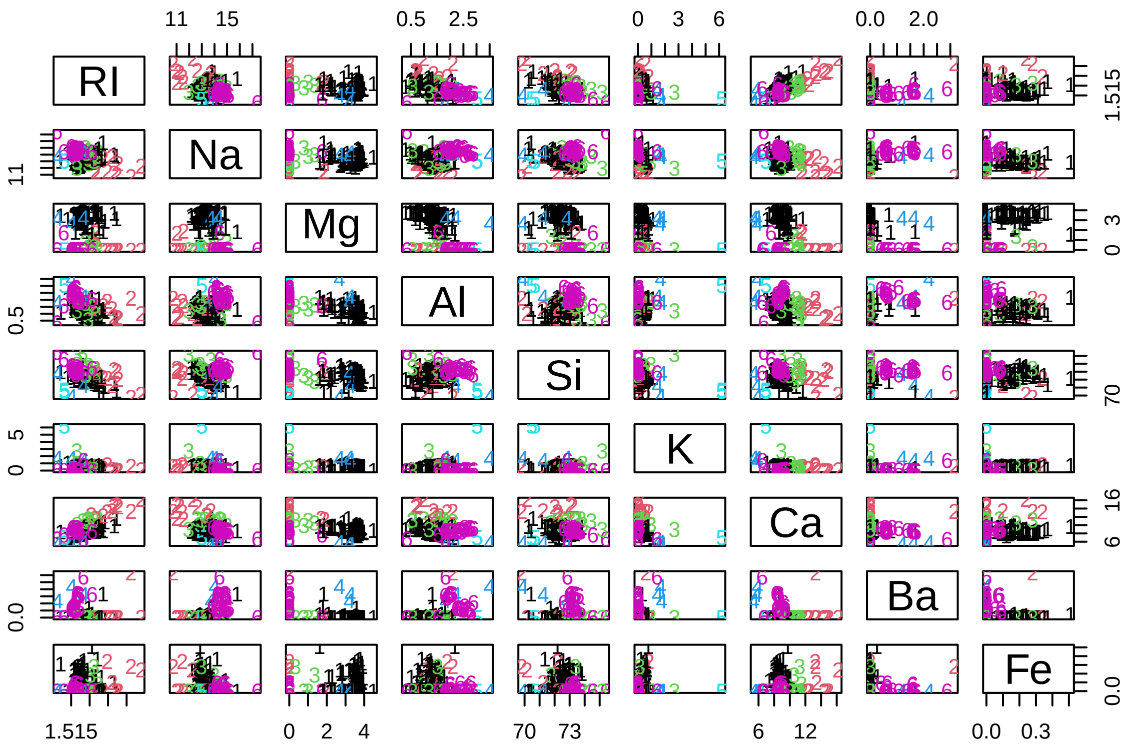
Complete Method



```
plot(data, col=complete.cut,pch = clusym[complete.cut])
```



```r
pairs(data, col=complete.cut,pch = clusym[complete.cut])
```



Gaussian mixture model

```
library(mclust)
```

```
## Warning: il pacchetto 'mclust' è stato creato con R versione 4.2.2
```

```
## Package 'mclust' version 6.0.0
## Type 'citation("mclust")' for citing this R package in publications.
```

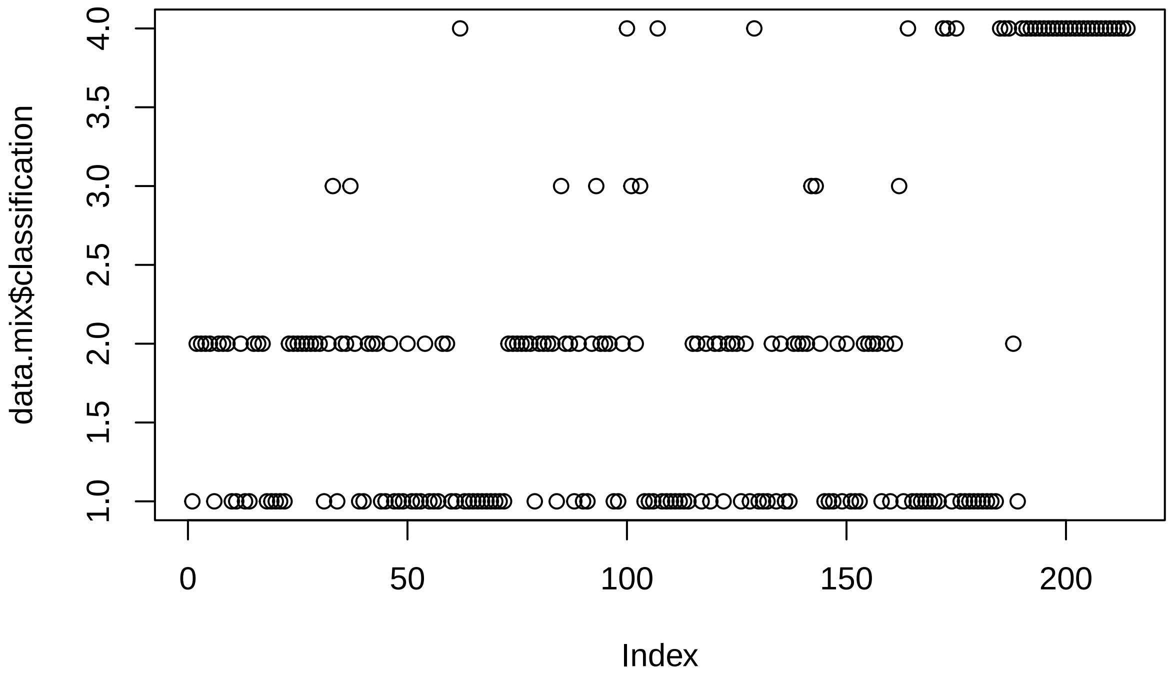
```
data.mix<-Mclust(data, G = 1:15)
data.mix$G
```

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

```
data.mix$classification
```

```
##   [1] 1 2 2 2 2 1 2 2 2 1 1 2 1 1 2 2 2 1 1 1 1 1 2 2 2 2 2 2 2 2 1 2 3 1 2 2 3
##  [38] 2 1 1 2 2 2 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 2 1 1 4 1 1 1 1 1 1 1 1 1 1 2 2
##  [75] 2 2 2 2 1 2 2 2 2 1 3 2 2 1 2 1 1 2 3 2 2 2 1 1 2 4 3 2 3 1 1 1 4 1 1 1 1
## [112] 1 1 1 2 2 1 2 1 2 2 1 2 2 2 1 2 1 4 1 1 1 2 1 2 1 1 2 2 2 2 3 3 2 1 1 1 2
## [149] 1 2 1 1 1 2 2 2 2 1 2 1 2 3 1 4 1 1 1 1 1 1 1 4 4 1 4 1 1 1 1 1 1 1 1 1 4
## [186] 4 4 2 1 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
```

```r
plot(data.mix$classification)
```



```r
data.mix$BIC
```

```
## Bayesian Information Criterion (BIC):
##          EII       VII          EEI          VEI       EVI       VVI       EEE
## 1  -4824.664 -4824.664 -1408.09385 -1408.09385 -1408.094 -1408.094  278.6458
## 2  -4157.524 -2752.020 -1018.48818    69.65928        NA        NA  518.3956
## 3  -4122.589 -2305.049 -1002.92952   445.11474        NA        NA  527.1063
## 4  -3892.680 -2155.514  -998.26583   630.83126        NA        NA  625.1669
## 5  -3206.680 -1836.309  -644.90885   805.00239        NA        NA  678.6716
## 6  -3046.679 -1752.675  -544.34882  1064.80359        NA        NA  825.2931
## 7  -2755.511 -1710.545  -204.22424  1310.82635        NA        NA  842.3792
## 8  -2774.714 -1604.881  -192.33931  1404.58412        NA        NA  876.4686
## 9  -2678.319 -1495.770    26.00282  1465.10417        NA        NA  895.5712
## 10 -2708.556 -1423.125   -14.48913  1456.46032        NA        NA 1106.3095
## 11 -2730.411 -1344.641    85.31256  1610.71247        NA        NA 1174.0798
## 12 -2653.694 -1355.110    31.65291  1661.71836        NA        NA 1128.1601
## 13 -2701.924 -1261.589   305.79033  1695.08196        NA        NA 1171.7879
## 14 -2728.441 -1197.676   303.53618  1726.85340        NA        NA 1203.6700
## 15 -2782.107 -1170.738   249.87609  1728.39403        NA        NA 1157.5132
##          VEE       EVE       VVE       EEV       VEV      EVV      VVV
## 1   278.6458  278.6458  278.6458  278.6458  278.6458 278.6458 278.6458
## 2  1316.4285 1279.1744 1248.7575 1748.5017 3066.7340       NA       NA
## 3  1432.4048        NA 1873.5898 1157.2659 4028.5616       NA       NA
## 4  1537.3672        NA        NA 1185.3919 4069.1406       NA       NA
## 5         NA        NA        NA 2219.7630        NA       NA       NA
## 6         NA        NA        NA 1988.5219        NA       NA       NA
## 7         NA        NA        NA 1843.5406        NA       NA       NA
## 8         NA        NA        NA        NA        NA       NA       NA
## 9         NA        NA        NA        NA        NA       NA       NA
## 10        NA        NA        NA        NA        NA       NA       NA
## 11        NA        NA        NA        NA        NA       NA       NA
## 12        NA        NA        NA        NA        NA       NA       NA
## 13        NA        NA        NA        NA        NA       NA       NA
## 14        NA        NA        NA        NA        NA       NA       NA
## 15        NA        NA        NA        NA        NA       NA       NA
##
## Top 3 models based on the BIC criterion:
##    VEV,4    VEV,3    VEV,2
## 4069.141 4028.562 3066.734
```

```
summary(data.mix$BIC)
```

```
## Best BIC values:
##             VEV,4      VEV,3     VEV,2
## BIC      4069.141 4028.56162  3066.734
## BIC diff    0.000  -40.57902 -1002.407
```

The Gaussian mixture model method shows that the best number of clusters is 4. All of them are of the type VEV and that means that they are ellipsoidal with equal shape and different volume and orientation.

From the classification output we can see that the first and the second clusters contain a lot of units while the others no. Specially the third group is characterized by very few units,

From the BIC output we can see that the first three best model are both VEV and that the fourth is the best.

Compare the clusterings and comment on how meaningful and useful you think they are. Select one clustering that you prefer. Discuss in particular whether the Gaussian mixture is a good method for these data in your view, and what might be potential problems with it. Produce at least one visualisation each for at least two clusterings. Interpret the clusters of the chosen clustering (you can use all given variables and your visualisations).

```
adjustedRandIndex(data.mix$classification,cluster.k$cluster)
```

```
## [1] 0.1456935
```

```
adjustedRandIndex(data.mix$classification,complete.cut)
```

```
## [1] 0.1879761
```

```
adjustedRandIndex(cluster.k$cluster,complete.cut)
```

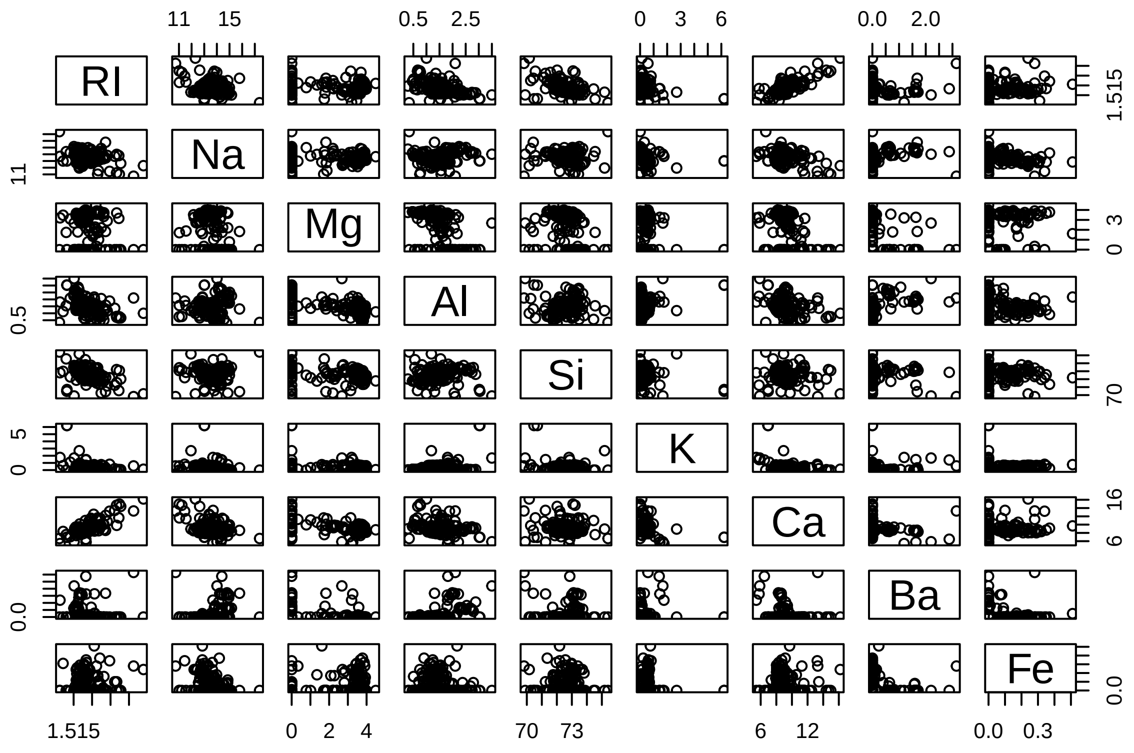
```
## [1] 0.2110239
```

All ARI values are very low and so very bad. The higher is the one obtained by comparing the clustering derived by the kmeans method and the one given by the hierarchical complete method. This means that they provide a similar clustering with respect to the mixture model criterion.

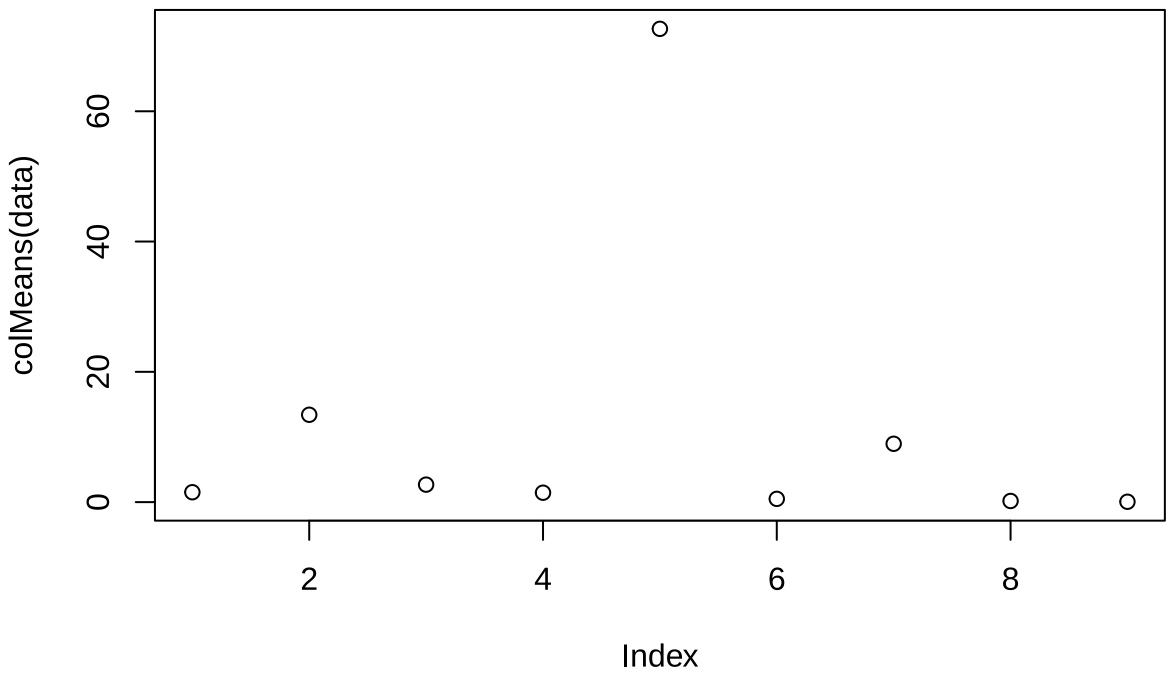
The kmeans method define the best k = 13 but from the plot we can see that 7 is a good value. The complete method shows k=6 so in this case they are similar. While the mixture model detects only 4 clusters. The mixture model method does not provides a good clustering for this dataset because it starts from a gaussianity assumption and thanks to the plot we can see that data are not normal distributed since they not assume a spherical form.

Comment on other aspects of the data set that you find out as far as you think they could be relevant.

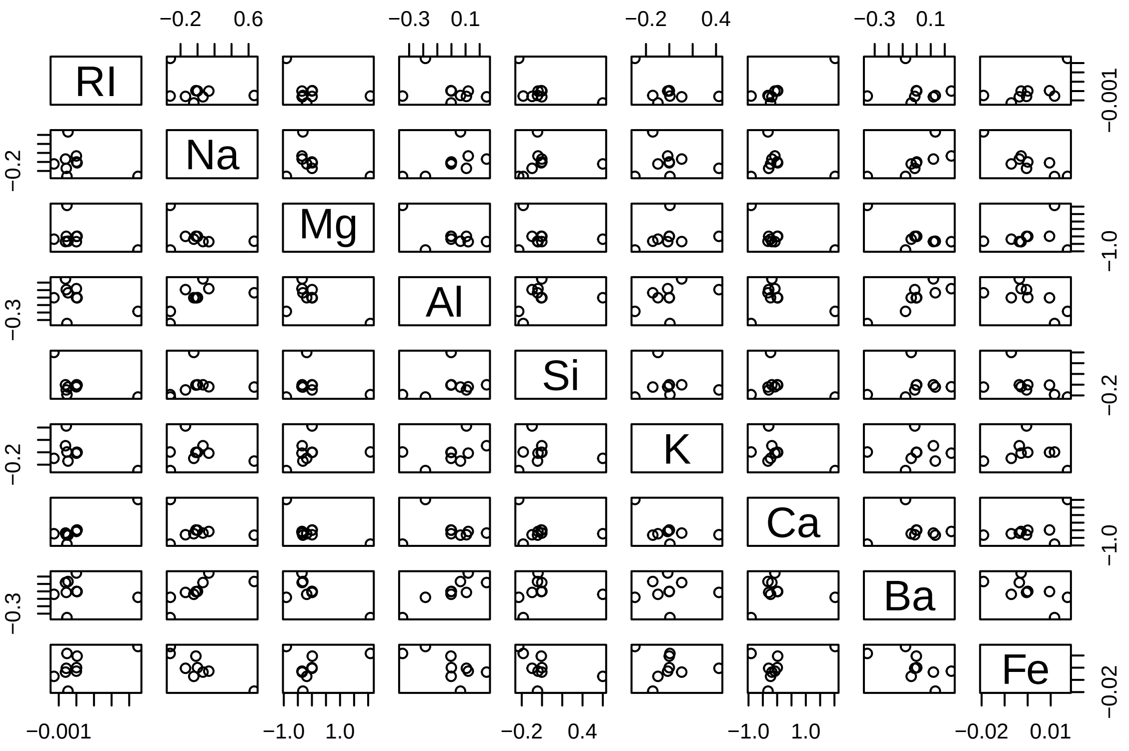
```
pairs(data)
```



```
View(data)

plot(colMeans(data))
```



```
pairs(var(data))
```



All variables seem to have very small variances and all variables' means are similar excepted for the sodium

(3)

(a) Summarize in your own words what the DBSCAN method does.

The DBSCAN is a clustering algorithm based on the density of clusters in order to discover clusters of arbitrary shape and detect ouliers. DBSCAN is efficient even for large spatial datasets. The main idea is that a point belongs to a cluster if it is quite near to a lot of points of this cluster. The key parameters are eps that specifies the neighborhoods, and minPts so the minimum number of data points to define a cluster. From these the points can be classified in three different categories: core points, if there are at least minPts number of points in its surrounding area defined by a radius; border points that are reachable from a core point and there are less than minPts points; outliers if they are not reachable from any core points. Both minPts and eps are defined and a starting point is selected from its surrounding area. If there are at least minPts point in the surrounding this point is assigned as core point, otherwise as outlier. Then we can choose a random point among points that have not been considered before and repeat until all points have been considered. DBSCAN then allows us to separate high density cluster and low-density ones.

(b) What are the advantages, according to the authors, of their DBSCAN method compared with other clustering methods, particularly those that you already know? Do you think that the authors' arguments are convincing?

The DBSCAN method is more valid in the theory than in the practice, it works well for large spatial datasets but only for no more than 3 dimensions. Its biggest advantage is that it does not consider the outliers while in other methods they affect the composition of the clusters. It also provides good clustering when the clusters have strange forms. Here are not necessary that all requirement for other clustering methods, only minPts and eps as input parameters is required but a good value for eps is difficoult to find out when clusters are different in terms of density Hierarchical and partitioning methods work well with normal distribution and points disposed in a elliptical or spherical forms. Partitioning method requires the value k (number of clusters) in input. Hierarchical does not need it but a termination condition to stop the algorithm is needed as well. The paper explains the method in detail and it seems to be very powerful. The property to be uncontaminated by the outliers leads to obtain important results. Also is very useful to have a method that can bypass problems related to the building of classical methods.

(c) Find out how to run DBSCAN in R and apply it to the Glass data from question 1 (you may also try it out on other datasets). You will need to make some decisions (and/or experiments) about tuning parameters. Comment on the results.

```r
library(dbscan)
```

```
## Warning: il pacchetto 'dbscan' è stato creato con R versione 4.2.2
```

```
##
## Caricamento pacchetto: 'dbscan'
```

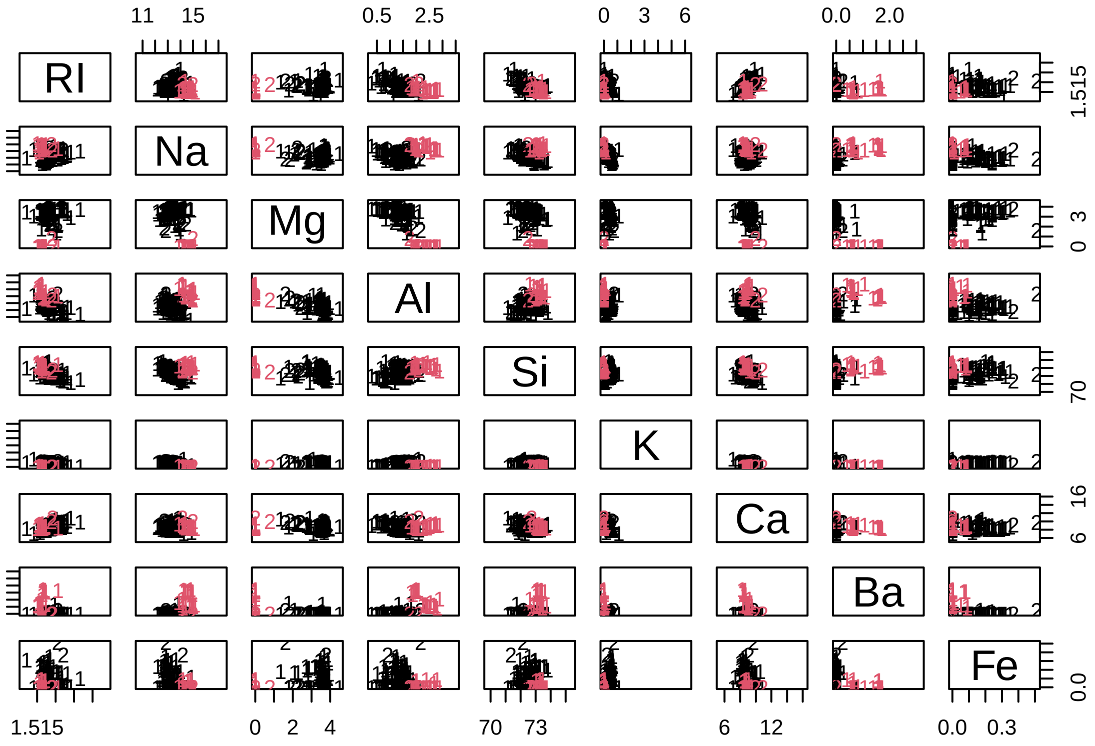
```
## Il seguente oggetto è mascherato da 'package:fpc':
##
##     dbscan
```

```
## Il seguente oggetto è mascherato da 'package:stats':
##
##     as.dendrogram
```

```r
set.seed(1234)
eps <- seq(1,2,0.1)
pts <- c(9:15)
sil.scores <- NULL
pts.list <- NULL
eps.list <- NULL
i = 0

sil.scores <- NULL
temp <- dist(data, method = 'euclidean')

for(eps in eps){
  for(pts in pts){
    i = i + 1
    scan <- dbscan(data, eps = eps, minPts = pts)
    pts.list[i] <- pts
    eps.list[i] <- eps
    sil.scores[i] <- summary(silhouette(scan$cluster, dist = temp))$avg.width
  }
}

dbg <- dbscan(data, eps = eps.list[which.max(sil.scores)], minPts = pts.list[which.max(sil.scores)])
dbg$cluster
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0
## [112] 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 2 2 0 0
## [186] 0 0 1 0 0 1 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 0 2 2 2 2 2 2
```

```
pairs(data, col=dbg$cluster, pch=clusym[dbg$cluster])
```



The DBSCAN method returns 2 clusters composed the first by 160 and the second by 23 units. 31 points are considered as outliers and to they are defined by label 0. It is obvious that two clusters are not sufficient, it is impossible that all units can be classified in only two clusters so we can say that this plot is not spatial and the method doesn't work well.