Projet de Classification Automatique

Estelle Descout - Alison Hourdeau - Mélanie Petton

Contents

Introduction

Au cours de ce projet, nous étudierons le jeu de données “camera.csv” portant sur des modèles d’appareils photos avec leurs caractéristiques correspondantes.

Nous allons mettre en oeuvre et comparer différentes méthodes de partitionnement telles que la méthode des k-means, la méthode des k-médoides et la classification mixte. Pour chaque méthode, nous allons dans un premier temps sélectionner le nombre de classes optimal puis comparer la division en clusters (ou classes), ainsi que l’inertie totale, l’inertie intra-classe, l’inertie inter-classes et la part d’inertie expliquée.

1. Importation des packages et des données

Au cours de l'étude, nous aurons besoin des librairies suivantes :

```
library(knitr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(plyr)
```

```
## ------------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
library(xtable)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(cluster)
library(factoextra)
```

```
## Loading required package: ggplot2
```

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

```
library(dendextend)
```

```
##
## ---------------------
## Welcome to dendextend version 1.14.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
```

```
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ----------------------
```

```
##
## Attaching package: 'dendextend'
```

```
## The following object is masked from 'package:stats':
##
##     cutree
```

```
library(ggplot2)
library(data.table)
```

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

```
## The following object is masked from 'package:dendextend':
##
##     set
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

Nous commençons par importer les données.

```
data = read.csv("Camera.csv", header=TRUE, sep = ";")
```

2. Analyse et traitement préalable

```
dim(data)
```

```
## [1] 1038   13
```

```
nb_indiv <- dim(data)[1]
nb_car <- dim(data)[2]
```

Le jeu de données "camera.csv" comporte 1038 lignes et 13 colonnes. Nous pouvons donc constater la présence de 1038 individus qui sont les modèles d'appareil photo. Pour chacun de ces modèles, 13 caractéristiques sont relevées, comme par exemple la date de sortie (release.date), les résolutions minimale et maximale (respectivement Low.resolution et Max.resolution), les dimensions (Dimensions) ou le prix (Price).

```
data_sans_NA = na.omit(data)
summary(data_sans_NA)
```

```
##                      Model       Release.date  Max.resolution Low.resolution
##  Agfa ePhoto 1280        :   1   Min.   :1994   Min.   :   0   Min.   :   0
##  Agfa ePhoto 1680        :   1   1st Qu.:2002   1st Qu.:2048   1st Qu.:1120
##  Agfa ePhoto CL18        :   1   Median :2004   Median :2560   Median :2048
##  Agfa ePhoto CL30        :   1   Mean   :2004   Mean   :2473   Mean   :1775
##  Agfa ePhoto CL30 Clik!:   1   3rd Qu.:2006   3rd Qu.:3072   3rd Qu.:2560
##  Agfa ePhoto CL45        :   1   Max.   :2007   Max.   :5616   Max.   :4992
##  (Other)                 :1030
##  Effective.pixels   Zoom.wide       Zoom.tele     Normal.focus.range
##  Min.   : 0.00    Min.   : 0.00   Min.   :  0.0   Min.   :  0.00
##  1st Qu.: 3.00    1st Qu.:35.00   1st Qu.: 96.0   1st Qu.: 30.00
##  Median : 4.00    Median :36.00   Median :108.0   Median : 50.00
##  Mean   : 4.59    Mean   :32.96   Mean   :121.5   Mean   : 44.13
##  3rd Qu.: 7.00    3rd Qu.:38.00   3rd Qu.:117.0   3rd Qu.: 60.00
##  Max.   :21.00    Max.   :52.00   Max.   :518.0   Max.   :120.00
##
##  Macro.focus.range Storage.included     Weight         Dimensions
##  Min.   : 0.000    Min.   :  0.00   Min.   :   0.0   Min.   :  0.0
##  1st Qu.: 3.000    1st Qu.:  8.00   1st Qu.: 180.0   1st Qu.: 92.0
##  Median : 6.000    Median : 16.00   Median : 226.0   Median :101.0
##  Mean   : 7.786    Mean   : 17.45   Mean   : 319.3   Mean   :105.4
##  3rd Qu.:10.000    3rd Qu.: 20.00   3rd Qu.: 350.0   3rd Qu.:115.0
##  Max.   :85.000    Max.   :450.00   Max.   :1860.0   Max.   :240.0
##
##      Price
##  Min.   :  14.0
##  1st Qu.: 149.0
##  Median : 199.0
##  Mean   : 457.9
##  3rd Qu.: 399.0
##  Max.   :7999.0
##
```

```
data=data_sans_NA
dim(data)
```

```
## [1] 1036   13
```

```
nb_indiv <- dim(data)[1]
nb_car <- dim(data)[2]
```

```
data.quanti = data[,cbind(3:13)]
stats_descr = round(sapply(data.quanti, each(min, max, mean, sd, var, median, IQR)),3)

xtable(stats_descr[,1:5],digits = 2)
```

	Max.resolution	Low.resolution	Effective.pixels	Zoom.wide	Zoom.tele
min	0.000	0.000	0.000	0.000	0.000
max	5616.000	4992.000	21.000	52.000	518.000
mean	2473.085	1774.859	4.590	32.956	121.544
sd	759.386	829.480	2.843	10.341	93.544
var	576666.393	688037.617	8.082	106.941	8750.517
median	2560.000	2048.000	4.000	36.000	108.000
IQR	1024.000	1440.000	4.000	3.000	21.000

```
xtable(stats_descr[,5:11],digits = 2)
```

	Zoom.tele	Normal.focus.range	Macro.focus.range	Storage.included	Weight	Dimensions	Price
min	0.000	0.000	0.000	0.000	0.000	0.000	14.000
max	518.000	120.000	85.000	450.000	1860.000	240.000	7999.000
mean	121.544	44.134	7.786	17.448	319.265	105.363	457.922
sd	93.544	24.164	8.104	27.441	260.410	24.263	761.089
var	8750.517	583.894	65.670	752.990	67813.440	588.682	579256.062
median	108.000	50.000	6.000	16.000	226.000	101.000	199.000
IQR	21.000	30.000	7.000	12.000	170.000	23.000	250.000

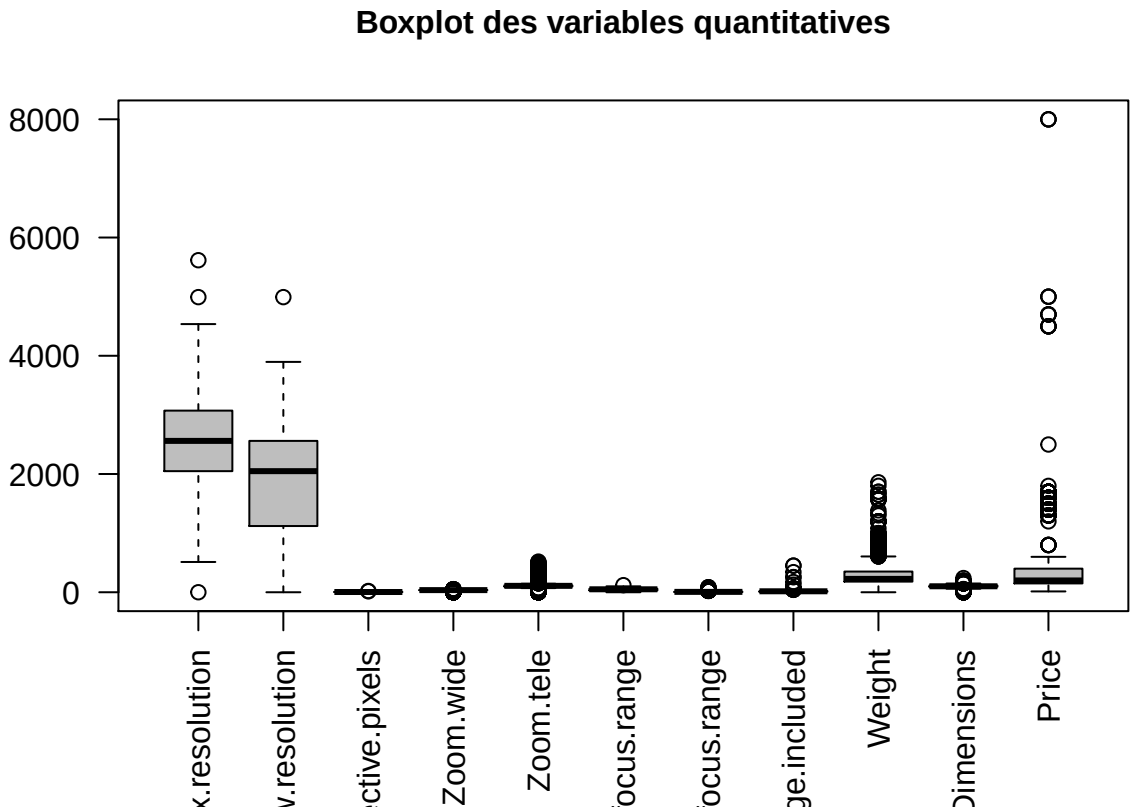
Grâce au tableau ci-dessus, nous pouvons obtenir plusieurs statistiques descriptives telles que les minimum et maximum, la moyenne, la variance ou encore la médiane. Par exemple, nous constatons que le prix minimum d'un modèle d'appareil photo est égal à 14 euros, le prix maximum est de 7999 euros. En moyenne, un appareil photo coûte 458 euros.

2.1. Graphiques

Nous allons réaliser de graphiques pour visualiser les données.

2.1.1. Boxplots

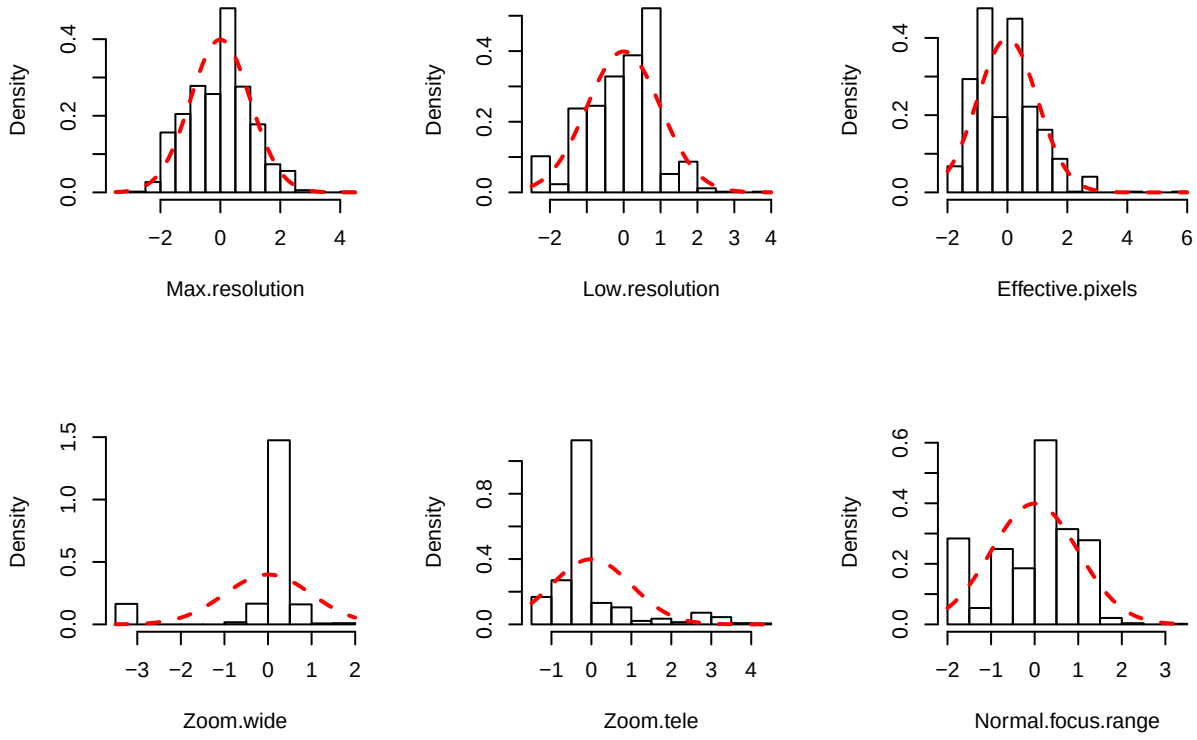
```
boxplot(data.quanti,main='Boxplot des variables quantitatives',cex.main=1,col='grey', las=2)
```



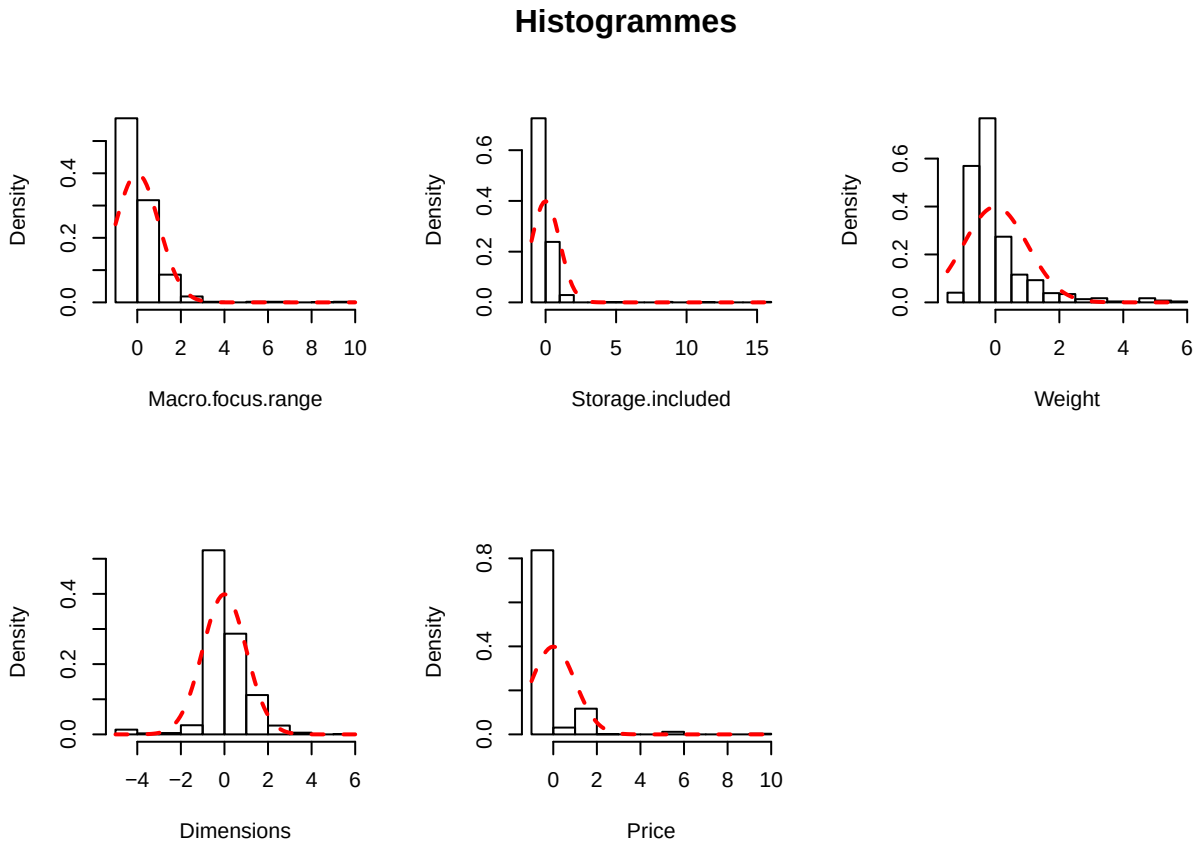
Grâce au boxplot ci-dessus, nous pouvons constater que les données des différentes variables sont peu dispersées, à l'exception des variables Max.resolution et Min.resolution pour lesquelles nous pouvons bien observer les différents indices de dispersion. De plus, nous remarquons que différents points sont situés en dehors des box-plots : il peut s'agir de valeurs aberrantes.

2.1.2 Histogrammes

```
#standardisation des données
data.quanti.norm <- scale(data.quanti,center=T,scale=T)

l = ncol(data.quanti.norm)
par(mfrow=c(2,3))
for (i in 1:l) {
hist(data.quanti.norm[,i],probability=TRUE,xlab=colnames(data.quanti.norm)[i],main='')

curve(dnorm(x), add=T, col="red", lwd=2, lty=2)
}
```

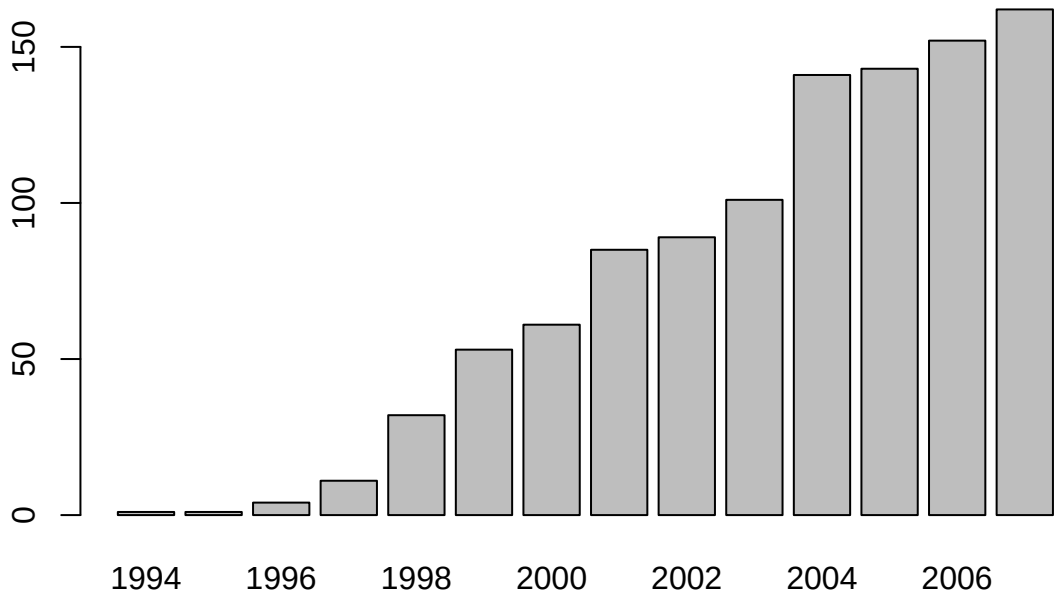



```
title("Histogrammes", outer=TRUE, line=-1,cex.main=1.5)
```



2.1.3. Graphique en barres

```
barplot(
  table(data$Release.date),
  main="Nombre d'appareils photos en fonction des années de sortie")
```

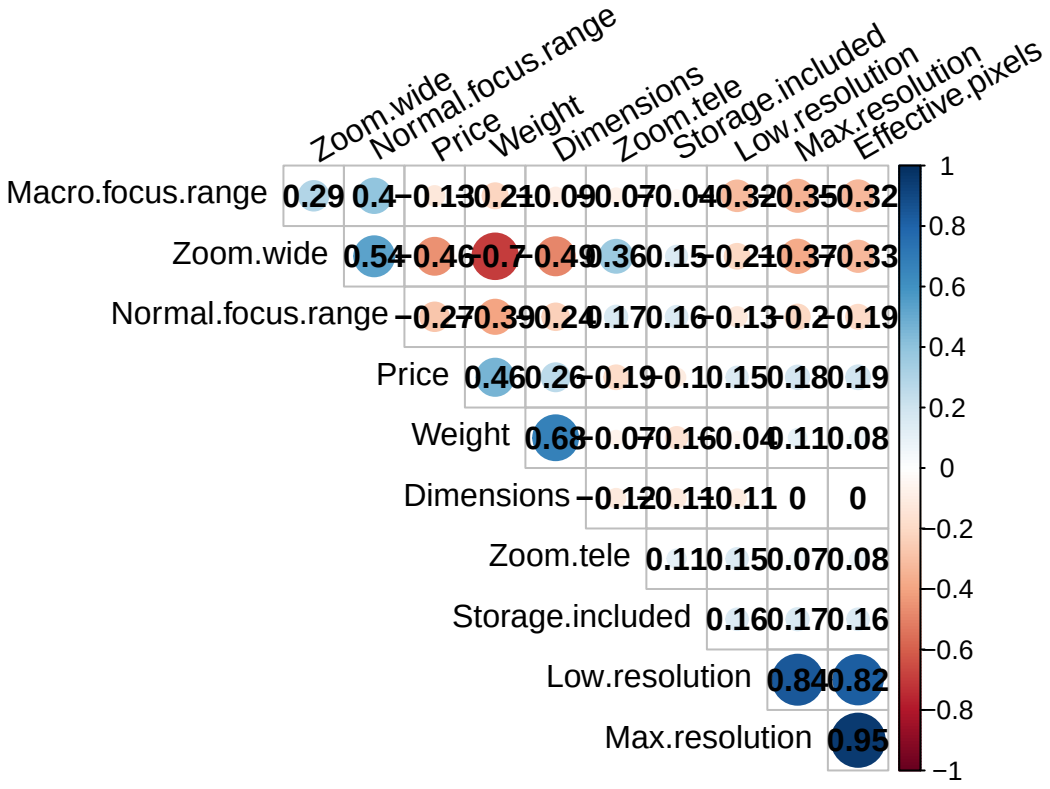


La parution de caméras n'a cessé d'augmenter. Nous remarquons une augmentation constante de leur nombre au cours des années. Même si depuis 2004 l'augmentation de leur nombre est plus faible, elle évolue tout de même positivement.

##2.2. Coefficients de corrélations

Nous pouvons calculer les coefficients de corrélation entre les différentes variables quantitatives, qui vont nous permettre d'étudier le lien entre celles-ci.

```
correlations = cor(data.quanti)


corrplot(
  correlations,
  method="circle",
  addCoef.col = "black",
  diag=FALSE,
  type="upper",
  order="hclust",
  tl.col="black",
  tl.srt=30)
```



Grâce à la matrice et au graphique ci-dessus, nous obtenons les coefficients de corrélation entre les variables quantitatives. Les variables Max.resolution et Effective.pixels sont fortement positivement corrélées (r=0.95), ainsi que les variables Low.resolution et Max.resolution (r=0.84) et Low.resolution et Effective.pixels (r=0.82). Cela signifie que lorsqu'une variable augmente, l'autre augmente également.

3. Implémentation de l'algorithme des centres mobiles : algorithme de Lloyd

Afin d'implémenter l'algorithme de Lloyd, nous avons créé plusieurs fonctions annexes.

D'abord, nous avons implémenté une fonction qui retourne le classe à laquelle appartient un individu. Pour cela, nous lui donnons en paramètres l'individu dont nous voulons connaître la classe, qui est donc un vecteur, ainsi que la matrice des barycentres que nous aurons calculée dans l'algorithme de Lloyd.

Nous initialisons un minimum *min* et le cluster *cluster* à NULL.

Nous parcourons ensuite la matrice des barycentres par ligne puisqu'une ligne correspond au barycentre d'une classe, nous ajoutons cette ligne au vecteur des individus. Nous appelons la matrice nouvellement obtenue *indiv_bary*.

Pour terminer, nous calculons la matrice des distances à partir de la matrice *indiv_bary*, nous ne retenons que la valeur de la deuxième ligne, première colonne car cette valeur correspond à la seule distance qui nous intéresse, à savoir la distance entre notre individu et le barycentre actuellement testé (barycentre de la classe i).

Enfin, si la distance obtenue est inférieure au minimum ou si le minimum est NULL, nous affectons la distance obtenue à la variable *min* et nous affectons sa classe à la variable cluster, correspondant au numéro de l'itération *i*.

Une fois la matrice des barycentres entièrement parcourue, nous aurons testé la distance entre l'individu et chacun des barycentres, nous aurons retenu la distance minimale et nous obtiendrons ainsi la classe de l'individu, que nous retournons.

```
cluster_by_indiv <- function(indiv, barycentres) {
  min <- NULL
  cluster <- NULL
  for(i in 1:nrow(barycentres)){
    indiv_bary <- rbind(indiv,barycentres[i,])
    distance <- as.matrix(dist(indiv_bary, method = "euclidean"))[2,1]
    if(!is.na(distance) && (min > distance || is.null(min))) {
      min <- distance
      cluster <- i
    }
  }
  return(cluster)
}
```

Nous avons ensuite décidé d'implémenter une fonction qui permet de savoir si deux matrices sont égales. Celle-ci nous servira pour savoir si les points ont changé de cluster entre deux itérations dans l'algorithme de Lloyd.

Nous passons en paramètre de la fonction deux matrices A et B. D'abord, nous testons l'égalité du nombre de colonnes entre A et B. Si elle n'est pas vérifiée, nous retournons false, de même pour le nombre de lignes. Enfin, nous parcourons les élements des deux matrices un à un et nous testons leur égalité. Si elle n'est pas vérifiée, nous retournons false. Après le parcours, si rien n'a été retourné, cela signifie que les matrices sont égales, nous retournons true.

```
matrices_egales <- function(A, B) {
  if( isTRUE(ncol(A) != ncol(B)) ) return(FALSE)
  if( isTRUE(nrow(A) != nrow(B)) ) return(FALSE)

  for( i in 1:nrow(A) ){
    for( j in 1:ncol(A) ) {
```

```
      if( A[i,j] != B[i,j] ) return(FALSE)
    }
  }
  return(TRUE)
}
```

La dernière fonction annexe que nous avons implémentée est une fonction qui indique si nous devons arrêter l'algorithme de Lloyd. Nous devons arrêter si le numéro de l'itération auquel nous sommes arrivées dépasse le nombre d'itérations maximum fixé ou si aucun individu ne change de classe. La condition "aucun individu ne change de classe" équivaut à la condition "l'inertie intra-classe ne diminue plus" ainsi qu'à la condition "le vecteur des barycentres est stable".

Nous passons en paramètres de la fonction : - un entier *i*, qui correspond au nombre d'itérations que nous avons déjà effectué; - un entier *max_iter*, qui correspond au nombre maximal d'itérations que nous avons fixé; - une matrice *data*, correspondant à la matrice des données actuelles, composée des individus et d'une colonne indiquant la classe de chacun pour cette itération; - une matrice *old_data*, correspondant à la matrice des données de la précédente itération, composée elle aussi des individus et d'une colonne indiquant leurs classes calculées à la précédente itération.

Comme précédemment indiqué, nous testons d'abord si le nombre d'itérations maximum est atteint, si c'est le cas nous renvoyons true. Sinon, nous renvoyons le résultat de la fonction **matrices_egales** à laquelle nous passons en paramètre les colonnes cluster de *data* et *old_data*. Ce résultat nous indiquera si les individus ont changé de classes entre les deux itérations. Si c'est le cas, nous renvoyons false et continons l'algorithme de Lloyd, sinon nous stoppons son déroulement.

```
haveToStop <- function(i, max_iter, data, old_data){
  if(i>=max_iter) return(TRUE)
  else return(matrices_egales(
                as.matrix(data$cluster),
                as.matrix(old_data$cluster)
              ))
}
```

À l'aide de ces trois fonctions, nous avons implémenté l'algorithme de Lloyd.Celui-ci prend en paramètres un entier k correspondant au nombre de classes souhaité et une matrice de données *data* :

```
algo_Lloyd <- function(k, data) {

  #En premier lieu, on vérifie la valeur de k :
  nb_indiv <- nrow(data)

  if(k<1) {
    stop("Le nombre de classes doit être supérieur à 1.")
  }

  if(k>nb_indiv) {
    stop("Le nombre de classes doit être inférieur au nombre d'individus.")
  }


  # On choisit aléatoirement et sans remise k points parmi les données data
  #Ces points seront les barycentres des k classes
  barycentres <- data[sample(1:nrow(data),k, replace = FALSE),]

  # i : nombre d'itérations, initialisé à 0
  i <- 0
```

```
  # max_iter : nombre d'itérations maximum, initialisé à 100
  max_iter <- 5


  # stop : booleen indiquant si l'on doit arrêter l'algorithme
  # On arrête si une des conditions suivantes est vraie :
  # min intra max inter
    # 1 - i >= max_iter
    # 2 - aucun indiv ne change de classe
    # 3 - inertie intra ne diminue plus
    # 4 - le vecteur des barycentres est stable
  stop <- FALSE

  #faire une fonction qui vérifie data cluster != null

  cluster <- c()
  for(d in 1:nrow(data)){
    cluster <- rbind(cluster,-1)
  }

  data <- cbind(data, cluster)

  while(!stop){
    old_data <- data

    data$cluster <- cluster

    if(i>0) {
      barycentres <- NULL
      for(j in 1:k){
        vect_barycentre <- colMeans(
                            subset(old_data[,-ncol(old_data)],
                                   old_data$cluster == j)
                            )
        barycentres <- rbind(barycentres, vect_barycentre)
      }
    }

    for(j in 1:nrow(data)){
      data[j,-1] <- cluster_by_indiv(data[j,-ncol(data)], barycentres)
    }

    stop <- haveToStop(i, max_iter, data, old_data)
    i <- i+1
  }
  return(data)
}
```

En premier lieu, nous testons la valeur de k. En effet, le nombre de classes doit être supérieur à 1 et inférieur au nombre d'individus. Si ce n'est pas le cas, nous stoppons directement l'algorithme en précisant la raison.

Ensuite, nous tirons aléatoirement les k barycentres de la première itération parmi les individus (lignes) de la matrice de données data.

Nous initialisons le nombre d'itérations i à 0 et le nombre d'itérations maximum à 20 afin que l'algorithme ne tourne pas infiniment s'il n'y a pas de convergence entre les clusters.

Nous initialisons le booléen stop à false, celui-ci permettra de savoir si une des conditions d'arrêt définie dans la fonction **haveToStop** est vérifiée.

Avant de commencer les itérations, nous initialisons également un vecteur *init_cluster*, composé d'une colonne et du même nombre de lignes que la matrice de données, ne contenant que des -1. Nous modifions la matrice des données en lui ajoutant ce vecteur.

Tant que stop n'est pas égale à true, nous itérons :

Nous stockons la matrice des données dans une variable *old_data* afin de modifier librement la matrice *data* et de garder les anciens clusters des individus pour le test d'arrêt. Nous réinitialisons la colonne cluster à l'aide du vecteur *cluster* initialisé au début de notre fonction. Si nous ne sommes pas à la première itération, nous recalculons la matrice des barycentres à partir des données de l'itération précédente, matrice *old_data*, afin de connaître les individus de chacune des classes. Pour se faire, pour chaque classe, nous utilisons la fonction R **colMeans** appliquée à la matrice *old_data* privée de la colonne cluster, en filtrant les données sur la classe correspondante. Une fois les barycentres calculés, nous parcourons les individus (lignes) de la matrice *data* et nous leur affectons leur classe à l'aide de notre fonction **cluster_by_indiv**.

Enfin, nous regardons si nous devons arrêter la fonction, à savoir si nous avons atteint le nombre d'itérations maximum ou si les individus n'ont pas changé de classes entre l'itération actuelle et la précédente. Nous incrémentons également i.

Lorsque la fonction est terminée, nous retournons la matrice *data*, contenant les individus et leurs classes.

4. Application de l'algorithme implémenté au jeu de données

Notre algorithme k-means étant implémenté, nous pouvons l'appliquer à nos données.

Nous devons déterminer le nombre de classes optimal pour cette méthode. Pour cela, nous utilisons la méthode du R^2. Nous avons dans un premier temps besoin de fonctions qui calculent les inerties intra, inter, totale et la part d'inertie expliquée.

4.1 Fonctions de calcul d'inertie

Pour calculer l'inertie, nous avons besoin de connaître la distance de l'ensemble des points de data avec leur barycentre. Voici une fonction qui retourne les barycentres d'un vecteur et une autre qui permet de retourner ce résultat sous forme de matrice :

```
barycentre<-function(vect){
  # apply permet d'appliquer ici la fonction mean aux colonnes de vect.
  return(apply(vect,2, mean)) }
```

```
distances<-function(vect){
  bar <- barycentre(vect) # Calcul du barycentre de l'ensemble de points.
  # On le rajoute à l'ensemble des points afin de calculer les distances
  # globales avec l'ensemble des points.
  vect.2 <- rbind(vect,bar) #
  z <- dist(vect.2,method = "euclidean")
  return (as.matrix(z))
}
```

Nous pouvons ainsi calculer l'inertie totale.

```
inertie_totale<-function(data){
  n <- nrow(data)
  res <- sum((distances(data)^2)[n+1,1:n])/n
  return(res)
}
```

De même pour l'inertie intra-classe :

```
inertie_intra<-function(data,k){
  n_var=ncol(data)
  n=nrow(data)
  inertie=c()
  effectifs=c()
  for(i in 1:k){
    donnees_cluster=filter(data,cluster==i)
    effectifs[i]=nrow(donnees_cluster)
    inertie[i]=inertie_totale(donnees_cluster[,1:n_var])
  }
  n <- nrow(data)

  inertie.intra<-(1/n)*sum(effectifs*inertie)
  return(inertie.intra)
}
```

Pour l'inertie inter-classe, voici la fonction que nous avons créée :

```
inertie_inter<-function(data,k){
  G=c()
  effectifs=c()
```

```
  ecarts=c()
  for(i in 1:k){
    donnees_cluster=filter(data,cluster==i)
    G=rbind(G,barycentre(donnees_cluster))
    effectifs[i]=nrow(donnees_cluster)
  }
  G=rbind(G,barycentre(data))
  nbar <- nrow(G)
  ecarts <-(as.matrix(dist(G))^2)[nbar,1:nbar-1]
  n<-nrow
  res = sum(ecarts*effectifs)/sum(effectifs)
  return(res)
}
```

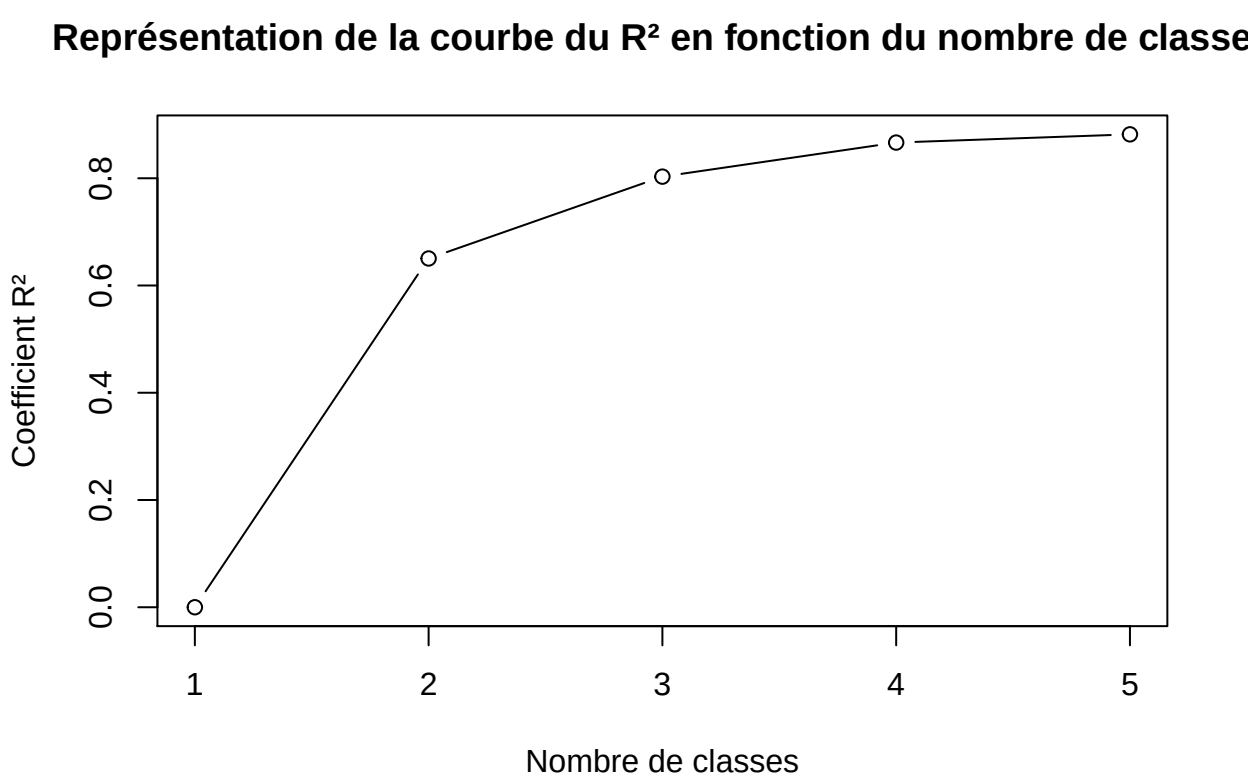
Et pour l'inertie expliquée, nous avons crée la fonction suivante :

```
inertie_expliquee<-function(data,k){
  inertie.intra=inertie_intra(data,k)
  inertie.totale=inertie_totale(data)
  return (round((1-(inertie.intra/inertie.totale))*100,2))
}
```

4.2 Nombre de classes et application

Nous allons pouvoir choisir le nombre de classes optimal avant d'appliquer notre algorithme kmeans implémenté au jeu de données.

```
result_kImpl <- c()

for (i in 1:5) {
  donnees_impl = algo_Lloyd(i,data.quanti)
  inter=inertie_inter(donnees_impl,i)
  tot=inertie_totale(donnees_impl)
  result_kImpl[i]=inter/tot
}

plot(1:5,
     result_kImpl,
     type='b',
     main =
        "Représentation de la courbe du R² en fonction du nombre de classes",
     xlab="Nombre de classes",
     ylab = "Coefficient R²")
```



Après l'étude du R^2 en fonction du nombre de classes, nous décidons de partitionner notre jeu de données en 3 classes avec la méthode du Kmeans implémentée.

Nous pouvons appliquer notre algorithme kMeans implémenté à notre jeu de données **data.quanti** en effectuant un découpage en trois classes.

```
dim(data.quanti)
```

```
## [1] 1036   11
```

```
dataLloyd=algo_Lloyd(3,data.quanti)
```

Nous pouvons appliquer nos différentes fonctions créées pour calculer les inerties suivantes :

```
inertie_intra(dataLloyd,3)
```

```
## [1] 113477.6
```

```
inertie_totale(dataLloyd)
```

```
## [1] 576117
```

```
inertie_inter(dataLloyd,3)
```

```
## [1] 462639.4
```

```
inertie_expliquee(dataLloyd,3)
```

```
## [1] 80.3
```

Nous pouvons calculer le pourcentage d'inertie expliquée lorsque nous appliquons notre kMeans implémenté à nos données.

Avec la méthode kMeans implémentée, nous obtenons un pourcentage d'inertie expliquée égal à 81,61 %. Nous pouvons donc conclure qu'une segmentation en trois classes est très optimal.

Nous obtenons la composition des trois classes à l'aide de la fonction suivante:

```
table(dataLloyd$cluster)
```

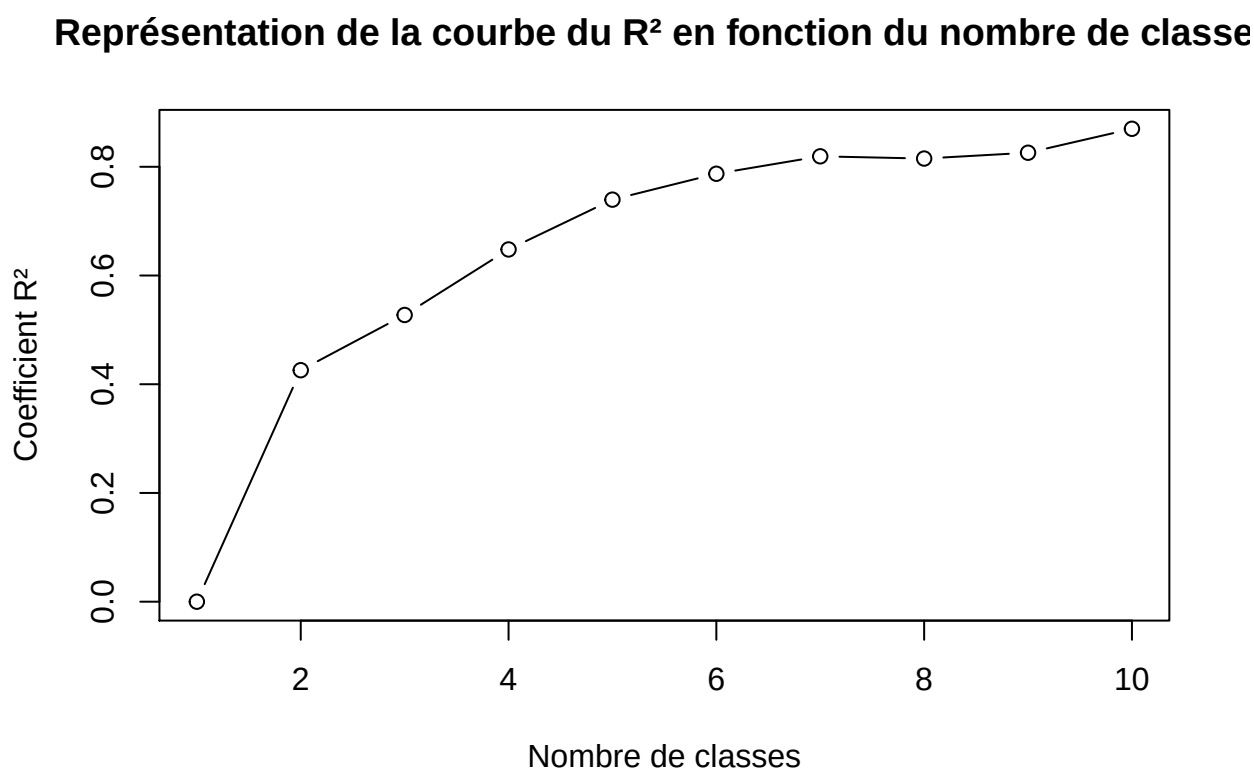
```
##
##   1   2   3
## 387 347 302
```

Nous pouvons noter que nous avons deux classes équilibrées en terme de nombre d'individus présents dans chacune d'entre elles.

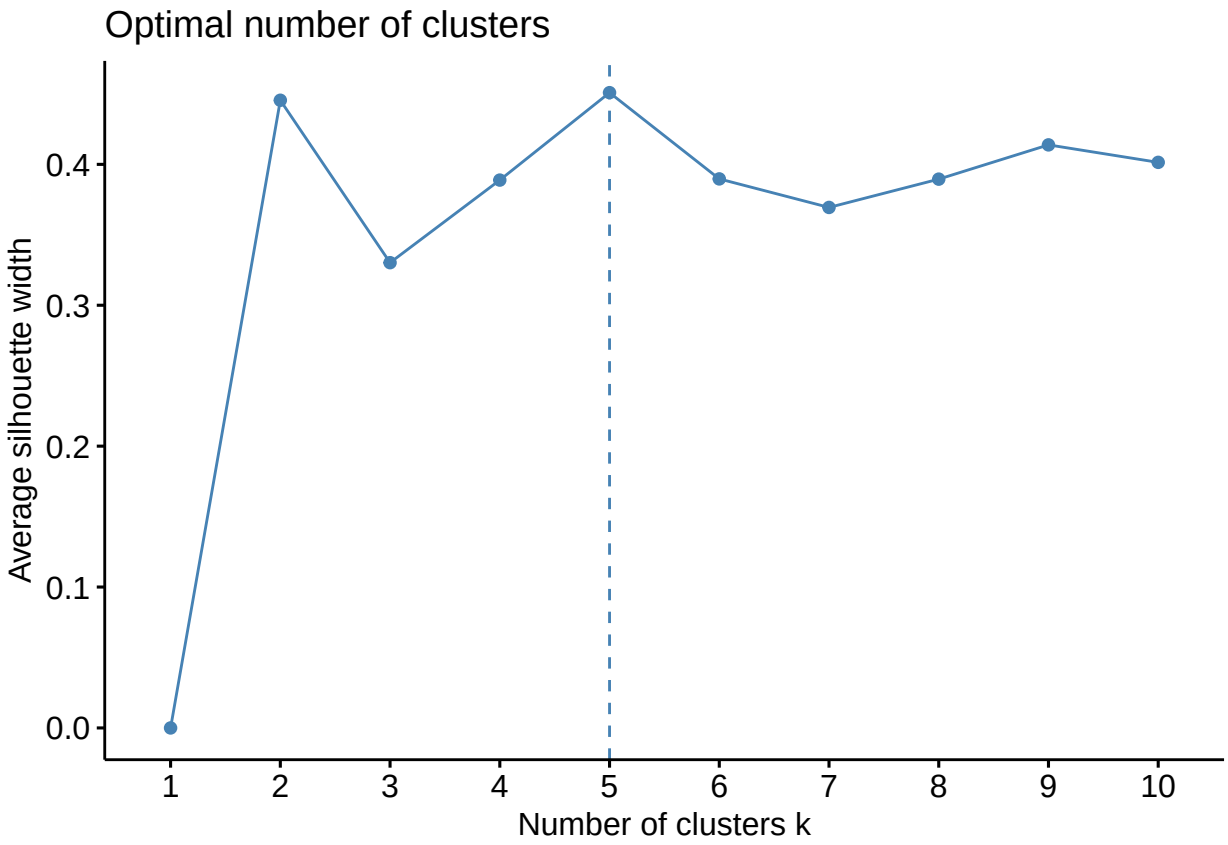
5. Segmentation à partir de la méthode des k-means de R

Après avoir réalisé notre fonction kMeans, nous allons étudier celle implémentée par R. Nous allons commencer par choisir le nombre de classes optimal.

```
result_kMeans <- c()

for (i in 1:10) {
  restmp=kmeans(data.quanti,centers=i,algorithm="MacQueen")
  result_kMeans[i]=restmp$betweenss/restmp$totss
}

plot(1:10,
     result_kMeans,
     type='b',
     main =
       "Représentation de la courbe du R² en fonction du nombre de classes",
     xlab="Nombre de classes",
     ylab = "Coefficient R²")
```




```
fviz_nbclust(data.quanti,kmeans,method="silhouette")
```



Nous voulons trouver le nombre de classes optimal à l'aide de la méthode du coude. Grâce à ce graphique, nous remarquons qu'à partir de 4 classes, l'adjonction d'un groupe supplémentaire n'augmente pas significativement la part d'inertie expliquée. Nous pouvons donc décider de garder 4 classes.

```
nb_class <- 4
kmeans.2 <- kmeans(data.quanti, centers=nb_class, algorithm=c("Lloyd"))
```

```
## Warning: did not converge in 10 iterations
```

Nous pouvons accéder aux différentes inerties grâce aux champs *totss*∗, ∗*tot.withinss* et *$betweenss*.

```
kmeans.2$totss #inertie totale
```

```
## [1] 1989922347
```

```
kmeans.2$tot.withinss #inertie intra
```

```
## [1] 700683795
```

```
kmeans.2$betweenss #inertie inter
```

```
## [1] 1289238552
```

Le pourcentage d'inertie expliquée avec 4 classes est de 64.788385 %. Nous obtenons ce résultat grâce à la ligne (between_SS / total_SS) issue de kmeans.2.

Ensuite, grâce à la ligne de commande ci-dessous, nous pouvons observer la composition des différentes classes :

```
kable(table(kmeans.2$cluster))
```

Var1	Freq
1	392
2	281
3	289
4	74

Nous pouvons constater que 281 appareils photo sont inclus dans la classe 1, 289 dans la classe 2, 393 dans la classe 3 et 73 dans la classe 4. Si nous voulons le détail, c'est à dire à quelle classe appartient chaque appareil photo, il suffit d'exécuter la ligne de commande kmeans$cluster.

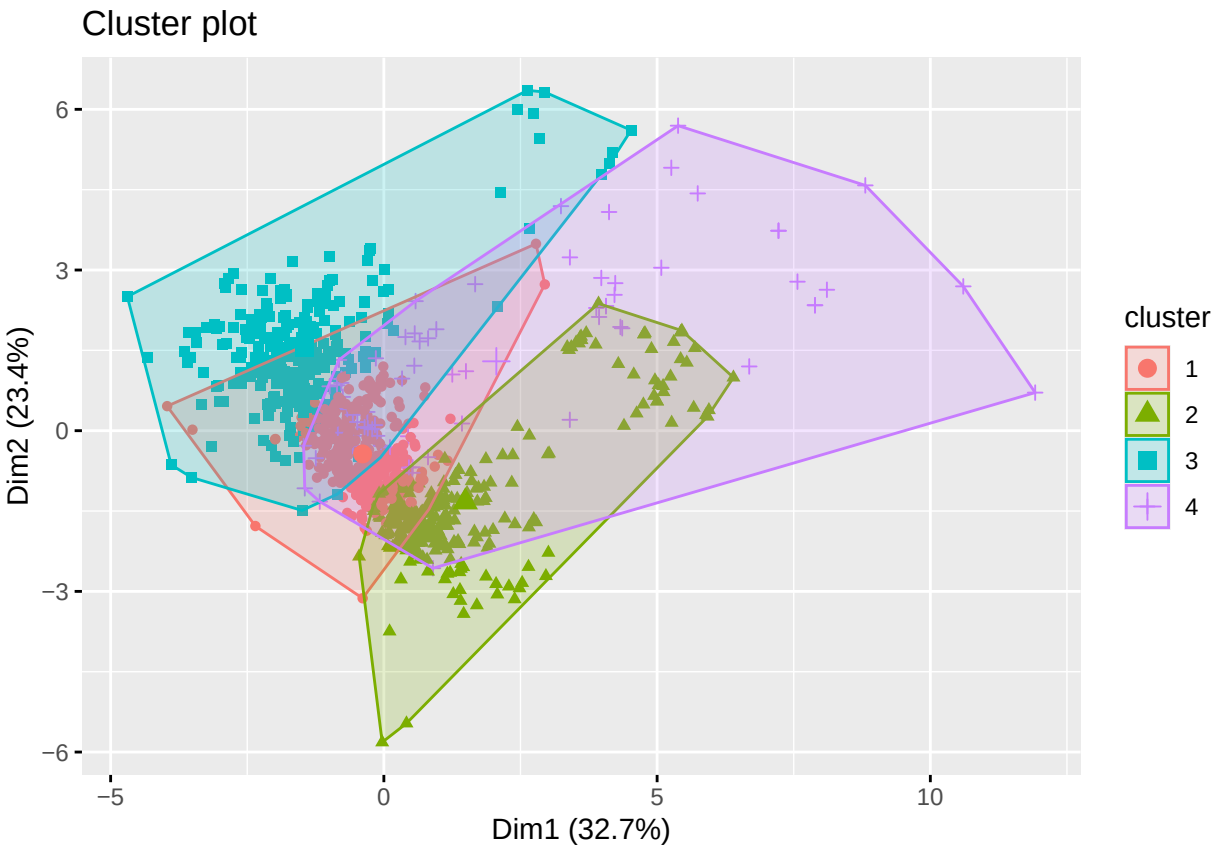
```
kmeans.2$centers
```

```
##   Max.resolution Low.resolution Effective.pixels Zoom.wide Zoom.tele
## 1       2464.921       1882.543         4.260204  35.76786 138.82653
## 2       3311.260       2676.815         7.907473  29.81139 136.54093
## 3       1608.301        699.910         1.643599  34.92388  97.77163
## 4       2710.865       1977.541         5.243243  22.31081  65.89189
##   Normal.focus.range Macro.focus.range Storage.included   Weight Dimensions
## 1           46.02296          7.627551         18.58929 241.0791   98.74617
## 2           40.45907          4.768683         24.27402 308.4591  104.63879
## 3           48.01038         11.259516         10.12111 368.4256  112.28547
## 4           32.94595          6.513514         14.09459 582.4865  116.13514
##       Price
## 1  220.0077
## 2  393.5907
## 3  357.5744
## 4 2354.4054
```

- La prmière classe sera composée des modèles d'appareil photo possédant des résolutions maximale et minimale faibles (moyennes respectivement environ égales à 3311.26 et 2676.82), des poids et dimensions moyens (308.46 et 104.64) ainsi qu'un prix moyen (393.59 euros).
- La deuxième classe sera composée des modèles d'appareil photo possédant des résolutions maximale et minimale élevées (moyennes respectivement environ égales à 1608.30 et 699.91), des poids et dimensions moyens (368.43 et 112.29) ainsi qu'un prix moyen (357.57 euros).
- La troisième classe sera donc composée des modèles d'appareil photo possédant des résolutions maximale et minimale moyennes (moyennes respectivement égales à 2463.86 et 1881.82), des poids faibles (240.93) et dimensions faibles (98.72) ainsi qu'un prix moyen (223.00 euros).
- La quatrième classe sera composée des modèles d'appareil photo possédant des résolutions maximale et minimale moyennes (moyennes respectivement environ égales à 2719.45 et 1977.54), des poids et dimensions élevées (587.93 et 116.48) ainsi qu'un prix très élevé (2367.49 euros).

Visualisation des résultats :

```
fviz_cluster (kmeans.2, data.quanti, geom="point")
```



Comparaison des résultats de la méthode kMeans de R avec ceux de notre méthode implémentée:

On remarque que les résultats sont différents de ceux de notre méthode implémentée lors de la précédente question. Ceci est probablement du à la définition de la fonction objectif. D'autre part, lorsque nous lançons la fonction kmeans de R, celle-ci nous indique que la convergence n'est pas atteinte malgré un grand nombre d'itérations. Ce manque de convergence implique des définitions de classes différentes.

Le pourcentage d'inertie expliquée est largement meilleur avec la méthode implémentée que celle de R (respectivement 81,3% et 64,8%).

6. Recherche documentaire et comparative des méthodes PAM et CLARA (méthodes de segmentation k-medoids)

(a) - Présentation des méthodes

Méthode PAM

La méthode PAM (Partitioning Around Medoid) consiste à considérer un représentant pour chaque classe, celui-ci étant censé être le plus central de la classe. L'idée de cet algorithme est de commencer avec un ensemble de k-médoides (observation d'une classe qui minimise la moyenne des distances aux autres observations de la classe) représentatifs parmi les observations du jeu de données puis échanger le rôle entre un objet médoide et un non médoide si cela permet de réduire la distance globale, ce qui revient à minimiser la fonction objectif. Celle-ci correspond à la somme des dissemblances de tous les objets à leur médoïde le plus proche. À chaque itération, un médoïd est mis en concurrence avec un autre individu aléatoire.

Après avoir trouvé un ensemble de k médoïdes, des clusters sont construits en attribuant chaque observation au médoïde le plus proche. Ensuite, chaque médoïde m sélectionné et chaque point de données non médoïde sont échangés et la fonction objectif est calculée.

Avantages de la méthode PAM :

- méthode moins sensible aux valeurs atypiques;
- méthode efficace pour des données de petite taille.

Inconvénients de la méthode PAM :

- méthode limitée par le nombre d'observations (matrices de dissimilarités à stocker) et en temps de calcul (algorithme en $O(n^2)$);
- méthode non adaptable pour une population importante d'objets.

Phases de construction de l'algorithme PAM :

1 Sélectionner k objets pour devenir les médoïdes, ou si ces objets ont été fournis, les utiliser comme médoïdes ; **2** Calculer la matrice de dissimilarité si elle n'a pas été fournie ; **3** Attribuer chaque objet à son médoïde le plus proche ;

Phase d'échange :

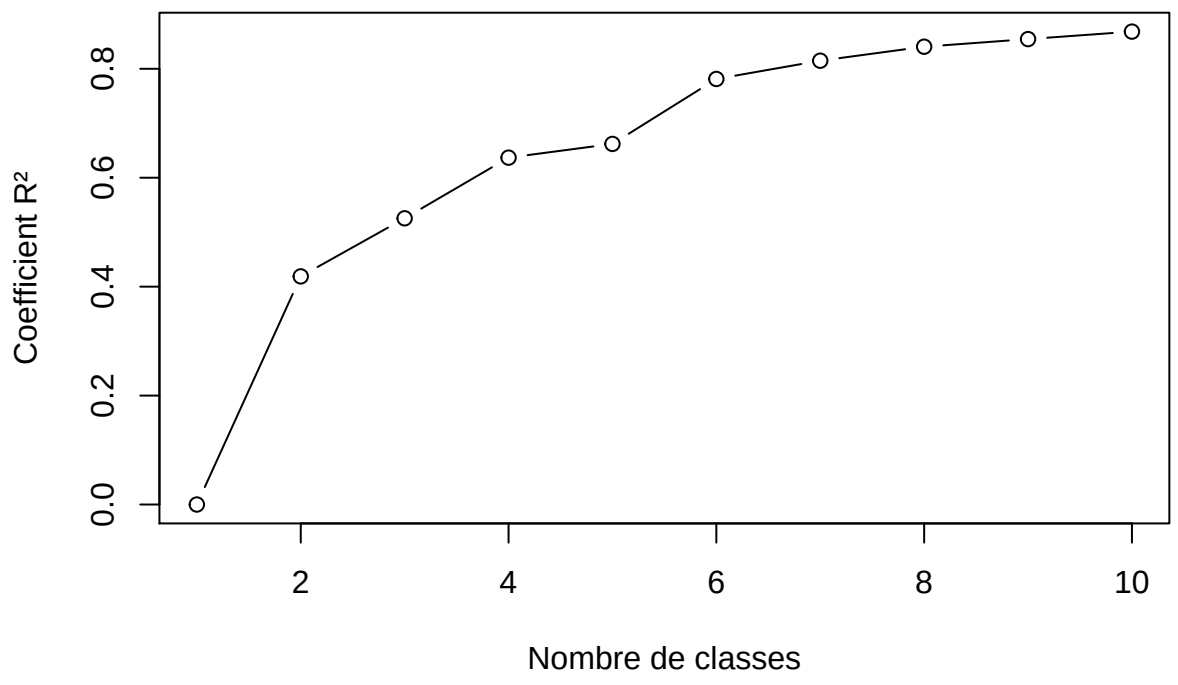
4. Pour chaque cluster, rechercher si l'un des objets du cluster diminue le coefficient de dissimilarité moyen ; si c'est le cas, sélectionner l'entité qui diminue le plus ce coefficient comme le médoïde pour ce cluster ; **5**Si au moins un médoïd a changé, passer à **(3)**, sinon terminer l'algorithme.

Réalisation de l'algorithme PAM sur R:

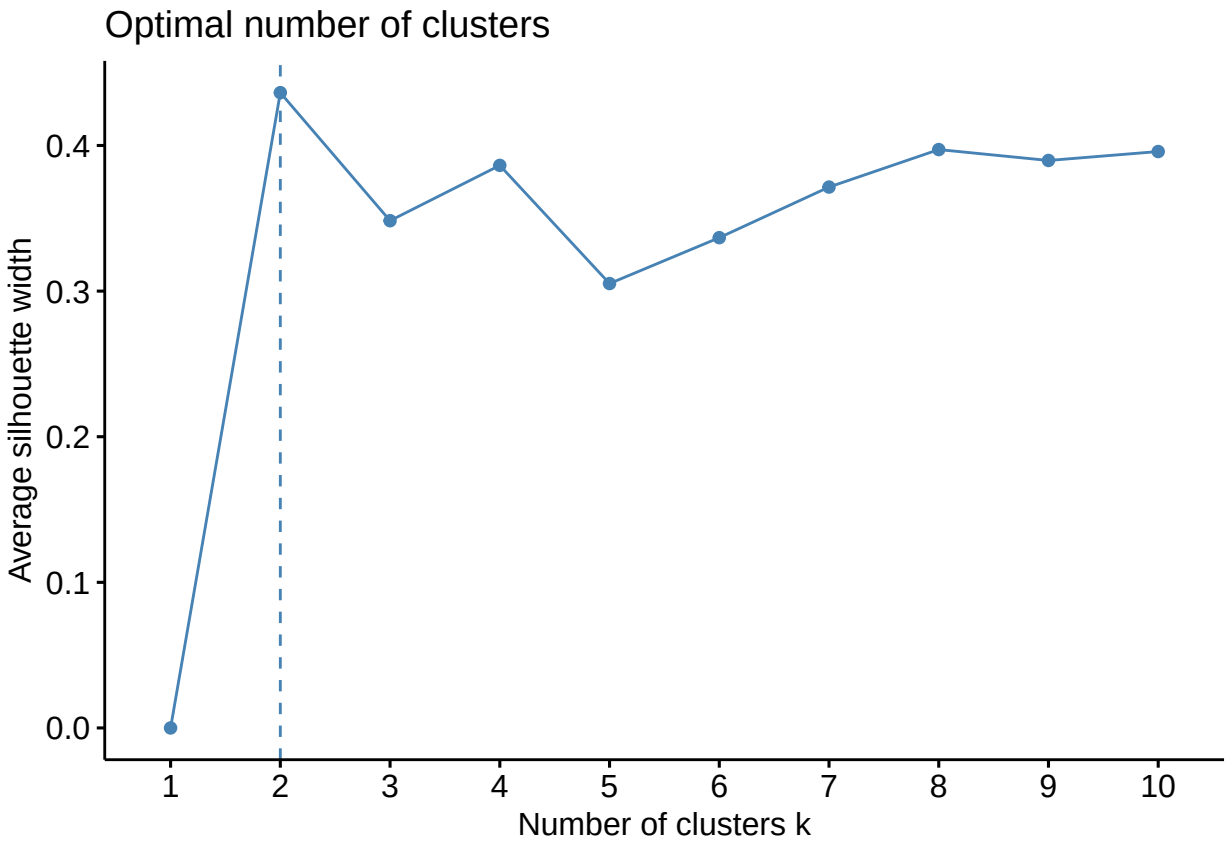
Nous appliquons la méthode *pam* de R, inclue dans le package cluster, dans laquelle nous rentrons le jeu de données data.quanti, le nombre de classes fixé, en spécifiant la distance euclidienne.

```
result_kPam <- c()

for (i in 1:10) {
  pam = pam(data.quanti, i, metric = "euclidean")
  donnees_pam=cbind(data.quanti,pam$clustering)
  setnames(donnees_pam, "pam$clustering", "cluster")
  inter=inertie_inter(donnees_pam,i)
  tot=inertie_totale(donnees_pam)
  result_kPam[i]=inter/tot
}

plot(1:10,
     result_kPam,
```

```
    type='b',
    main =
      "Représentation de la courbe du R² en fonction du nombre de classes",
    xlab="Nombre de classes",
    ylab = "Coefficient R²")
```




```
fviz_nbclust(data.quanti,cluster::pam,method = "silhouette")
```



Grâce aux graphiques ci-dessus, nous choisissons de couper le jeu de données en deux classes.

```
k=2
pam = pam(data.quanti, k, metric = "euclidean")

donnees_pam=cbind(data.quanti,pam$clustering)
setnames(donnees_pam, "pam$clustering", "cluster")
```

Nous pouvons utiliser les fonctions précédentes pour calculer les différentes inerties:

```
inertie_intra(donnees_pam,k)
```

```
## [1] 1116331
```

```
inertie_totale(donnees_pam)
```

```
## [1] 1920775
```

```
inertie_inter(donnees_pam,k)
```

```
## [1] 804443.3
```

```
inertie_expliquee(donnees_pam,k)
```

```
## [1] 41.88
```

Nous avons un pourcentage d'inertie expliquée égal à 41,88 % quand nous appliquons la fonction pam aux données, avec une segmentation en deux classes.

(b) - Composition des classes:

```
table(donnees_pam$cluster)
```

```
##
##   1   2
## 438 598
```

Grâce à la fonction table, nous pouvons obtenir le nombre d'individus dans chacune des classes.

```
pam$medoids # objets qui représentent les centres de classes.
```

```
##     Max.resolution Low.resolution Effective.pixels Zoom.wide Zoom.tele
## 479           1632           1024                1        38       114
## 130           2816           2304                6        38       114
##     Normal.focus.range Macro.focus.range Storage.included Weight Dimensions
## 479                  0                 0                8    295        108
## 130                 40                15                8    155         90
##     Price
## 479   229
## 130   249
```

Nous remarquons que la première classe est composée de modèles d'appareils photos de faibles résolutions (Max.resolution et Low.resolution moins élevées que dans la deuxième classe), avec un poids et des dimensions élevés.

La deuxième classe présente des modèles d'appareil photo avec des résolutions plus élevées que dans la première classe, un poids et une dimension faibles.

Avec la méthode PAM de R, nous obtenons un pourcentage d'inertie expliquée de 52.55% avec une segmentation en trois classes.

```
k=3
pam = pam(data.quanti, k, metric = "euclidean")

donnees_pam=cbind(data.quanti,pam$clustering)
setnames(donnees_pam, "pam$clustering", "cluster")
```

Nous pouvons utiliser les fonctions précédentes pour calculer les différentes inerties:

```
inertie_intra(donnees_pam,k)
```

```
## [1] 911448.4
```

```
inertie_totale(donnees_pam)
```

```
## [1] 1920775
```

```
inertie_inter(donnees_pam,k)
```

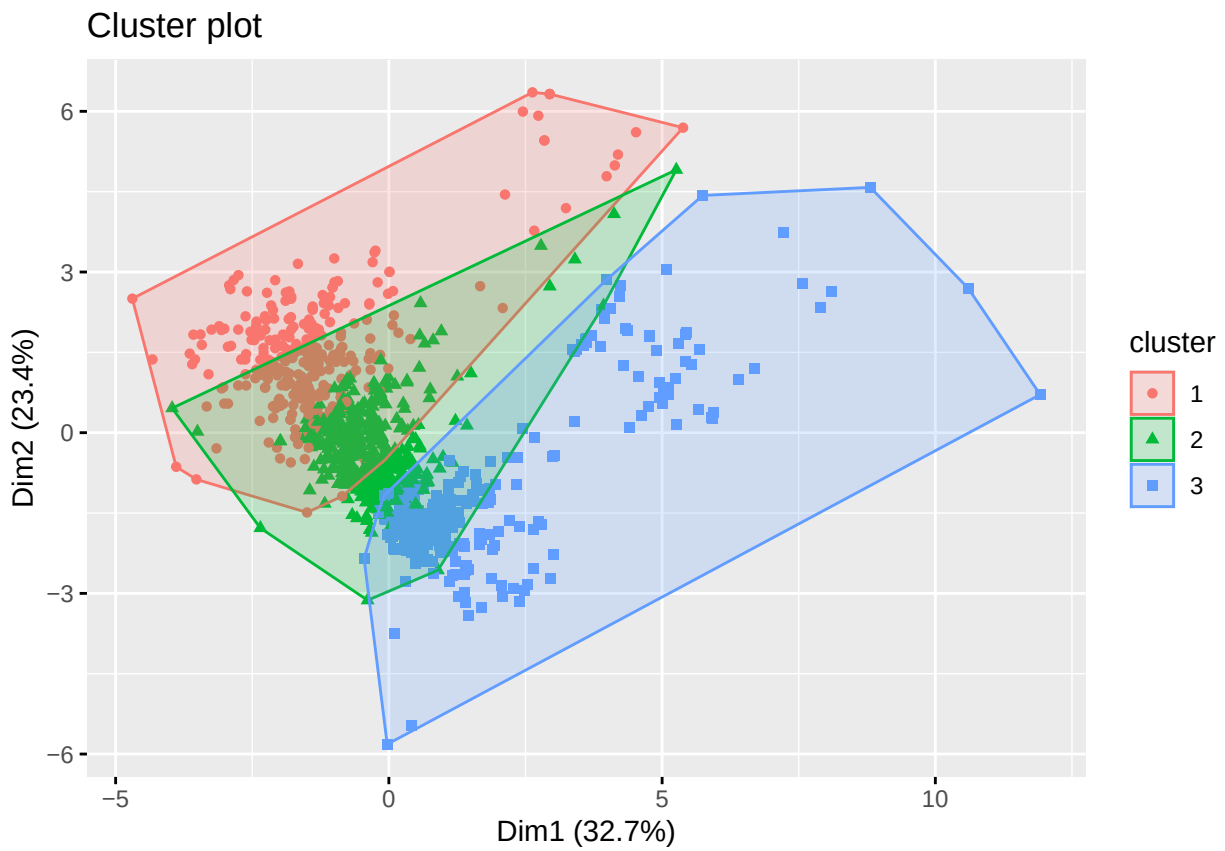
```
## [1] 1009327
```

```
inertie_expliquee(donnees_pam,k)
```

```
## [1] 52.55
```

Nous pouvons représenter graphiquement la visualisation des différents clusters :

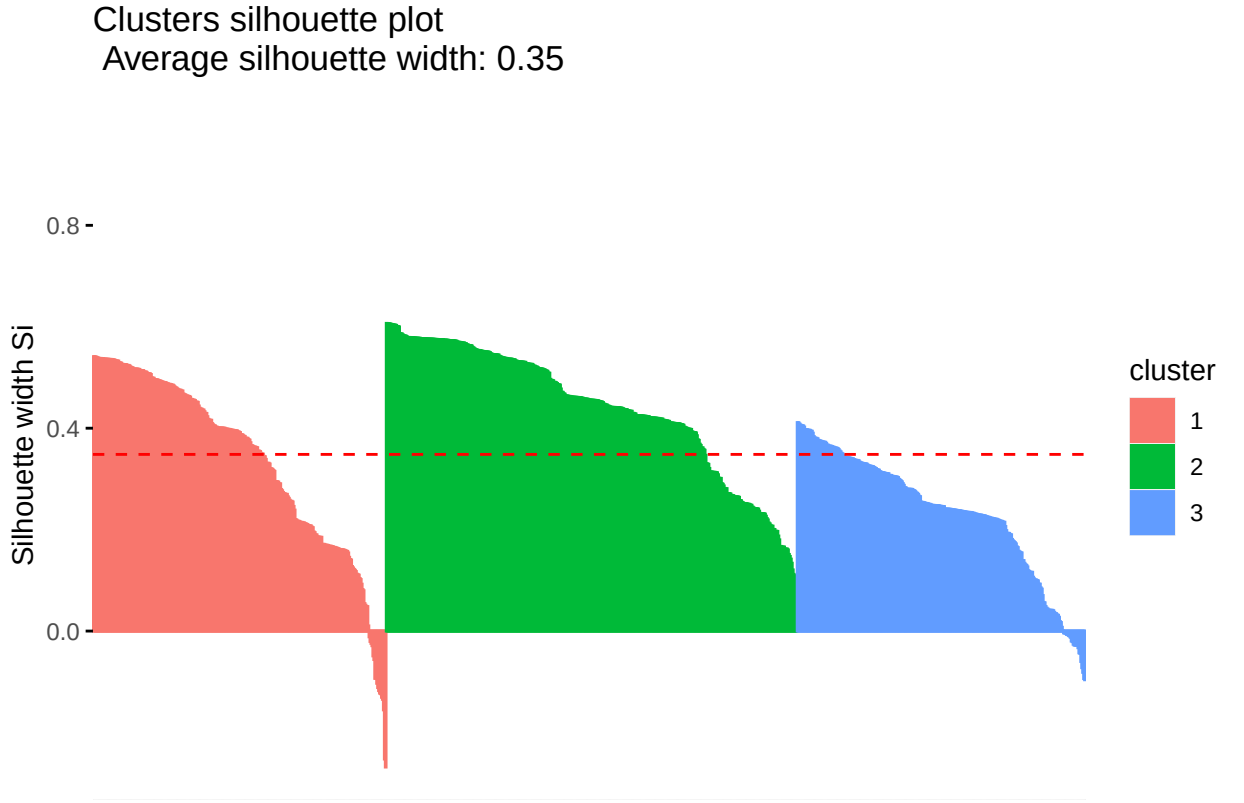
```
fviz_cluster (pam, data.quanti, geom="point")
```



Pour mesurer la qualité d'une partition d'un ensemble de données, nous allons calculer le coefficient de silhouette. Celui-ci désigne la différence entre la distance moyenne avec les points du même groupe que lui et la distance moyenne avec les points des autres groupes voisins. Si la différence est négative, le point est en moyenne plus proche du groupe voisin que du sien, ce qui signifie qu'il sera mal classé. Au contraire, si la différence est positive, le point est en moyenne plus proche de son groupe que du groupe voisin. Il sera donc bien classé.

```
fviz_silhouette(silhouette(pam))
```

```
##   cluster size ave.sil.width
## 1       1  306          0.33
## 2       2  429          0.44
## 3       3  301          0.23
```



Nous pouvons remarquer sur le graphique précédent que certains individus ont un coefficient de silhouette négatif dans la classe 2. Cela indique que ces individus ont mal été classés.

Méthode CLARA

La méthode CLARA (Clustering Large Applications) a été dévéloppée dans le but de réduire le coût de calcul de PAM. Cet algortihme travaille sur des échantillons au lieu de la population totale et leur applique à chaque fois la méthode PAM en retenant le meilleur résultat.Si l'échantillon est choisi d'une manière aléatoire, alors il représente bien tous les objets, donc les médoides sont similaires à ceux qui sont créés à partir de tous les objets. L'algorithme est, généralement, exécuté sur plusieurs échantillons pour obtenir le meilleur résultat.

Avantages de la méthode CLARA :

- possibilité de traiter de grandes bases.
- réduction du coût de calcul.

Inconvénients de la méthode CLARA :

-possibilité de ne pas atteindre la meilleure solution si l'objet qui serait le meilleur médoide n'apparait dans aucun échantillon; -l'algorithme est fortement dépendant de la taille et de la représentativité échantillons.

Phases de construction de l'algorithme CLARA :

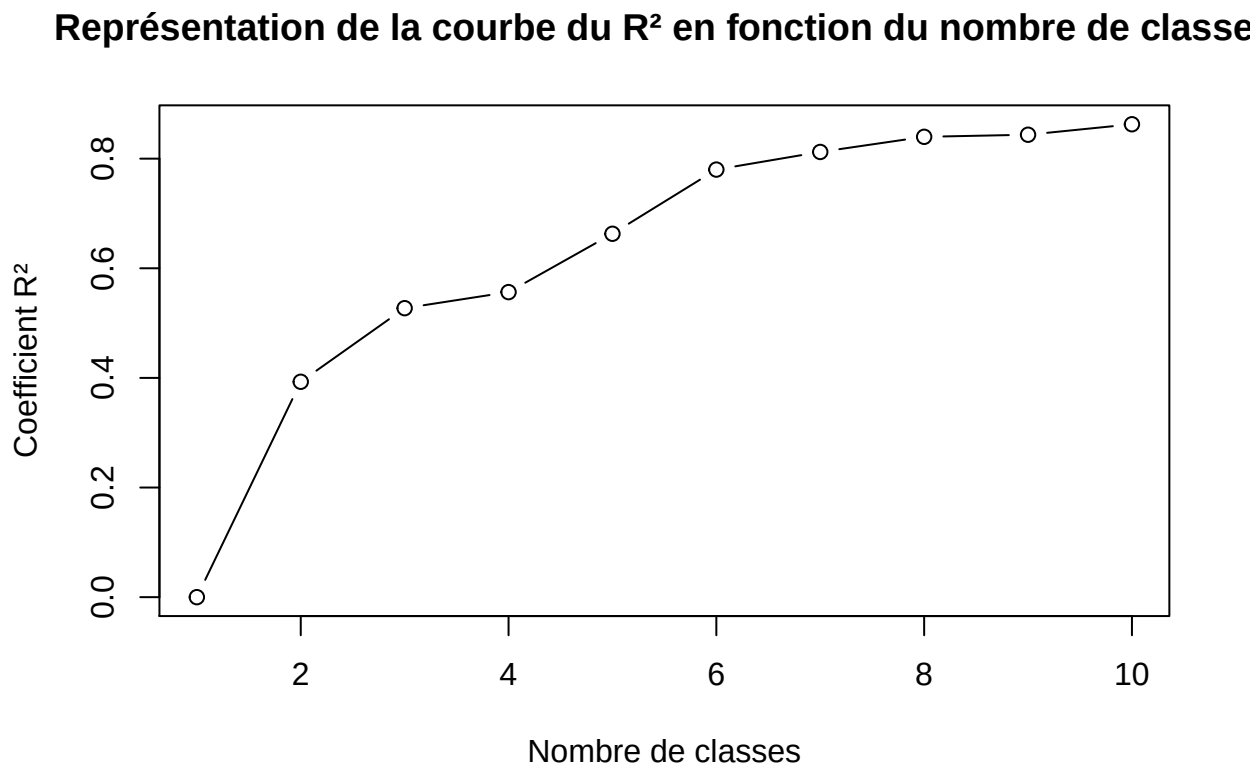
1 Créer aléatoirement, à partir de l'ensemble de données d'origine, plusieurs sous-ensembles de taille fixe (sampsize). **2** Calculer l'algorithme PAM sur chaque sous-ensemble et choisir les k objets représentatifs correspondants (medoids). Attribuer chaque observation de l'ensemble des données au médoïde le plus proche. **3** Calculer la moyenne (ou la somme) des dissemblances des observations à leur médoïde le plus proche. Ceci est utilisé comme une mesure de la qualité du clustering. **4** Conserver le sous-ensemble de données pour lequel la moyenne (ou somme) est minimale. Une analyse plus approfondie est effectuée sur la partition finale.

Réalisation de l'algorithme CLARA sur R:

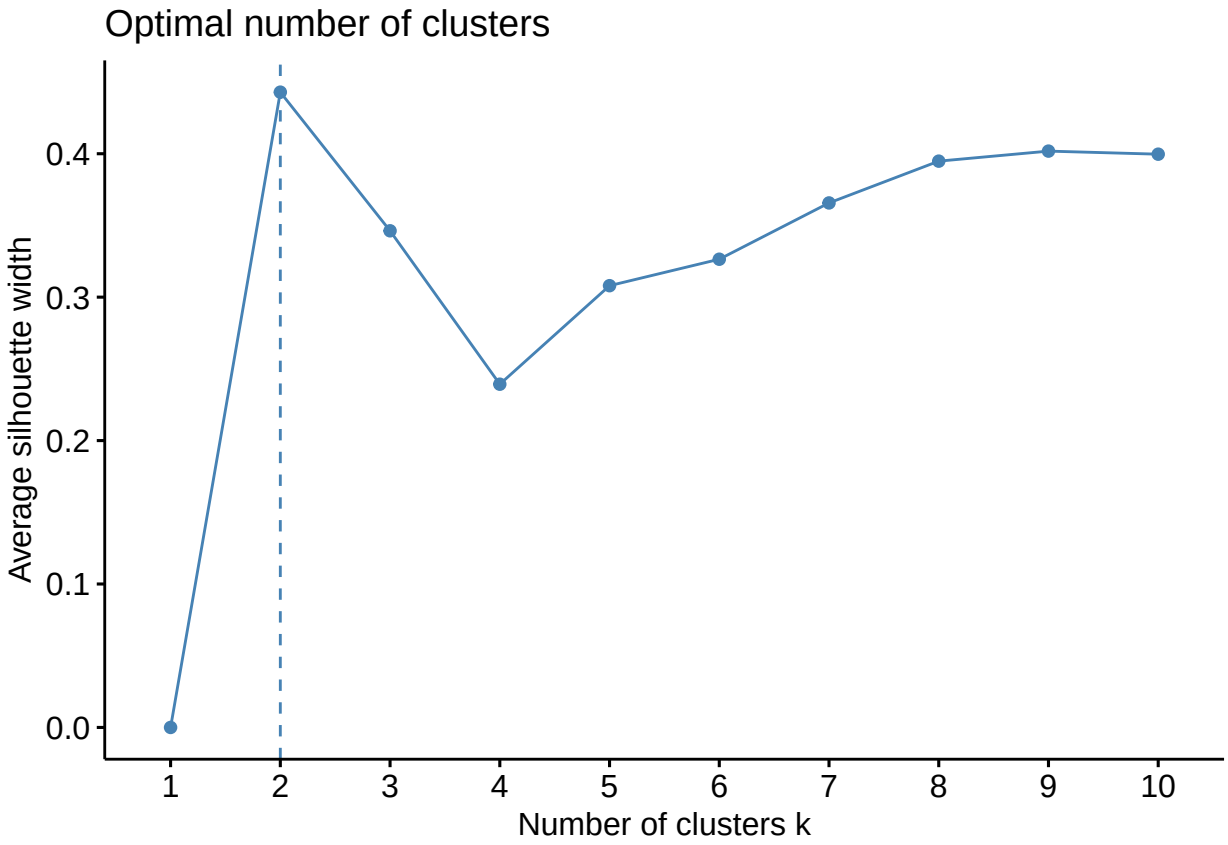
Pour effectuer la méthode CLARA sur R, nous utiliserons la fonction clara du package cluster. Nous procédons de la même manière que pour la méthode PAM : nous rentrons le jeu de données data.quanti, le nombre de classes fixé, en spécifiant la distance euclidienne.

Nous allons dans un premier temps choisir le nombre de classes nécessaires.

```
result_kClara <- c()

for (i in 1:10) {
  clara = clara(data.quanti, i, metric = "euclidean")
  donnees_clara=cbind(data.quanti,clara$clustering)
  setnames(donnees_clara, "clara$clustering", "cluster")
  inter=inertie_inter(donnees_clara,i)
  tot=inertie_totale(donnees_clara)
  result_kClara[i]=inter/tot
}

plot(1:10,
     result_kClara,
     type='b',
     main =
       "Représentation de la courbe du R² en fonction du nombre de classes",
     xlab="Nombre de classes",
     ylab = "Coefficient R²")
```



```
fviz_nbclust(data.quanti,cluster::clara,method="silhouette")
```



Après avoir observé les graphiques ci-dessus, nous pouvons décider de garder 2 classes. Nous pouvons donc appliquer la méthode CLARA sur notre jeu de données, avec une segmentation en deux classes.

```
k=2
clara = clara(data.quanti, k, metric = "euclidean") # samples par défaut : 5
donnees_clara=cbind(data.quanti,clara$clustering)
setnames(donnees_clara, "clara$clustering", "cluster")
```

Nous pouvons calculer les inerties en appelant les fonctions suivantes:

```
inertie_intra(donnees_clara,k)
```

```
## [1] 1166042
```

```
inertie_totale(donnees_clara)
```

```
## [1] 1920775
```

```
inertie_inter(donnees_clara,k)
```

```
## [1] 754732.5
```

```
inertie_expliquee(donnees_clara,k)
```

```
## [1] 39.29
```

Le pourcentage d'inertie expliquée en appliquant la méthode clara avec deux classes est de 39,29%.

Nous pouvons obtenir la composition des classes grâce à la fonction suivante, c'est à dire le nombre d'individus présents dans chaque classe.

```
table(donnees_clara$cluster)
```

```
##
##   1   2
## 304 732
```

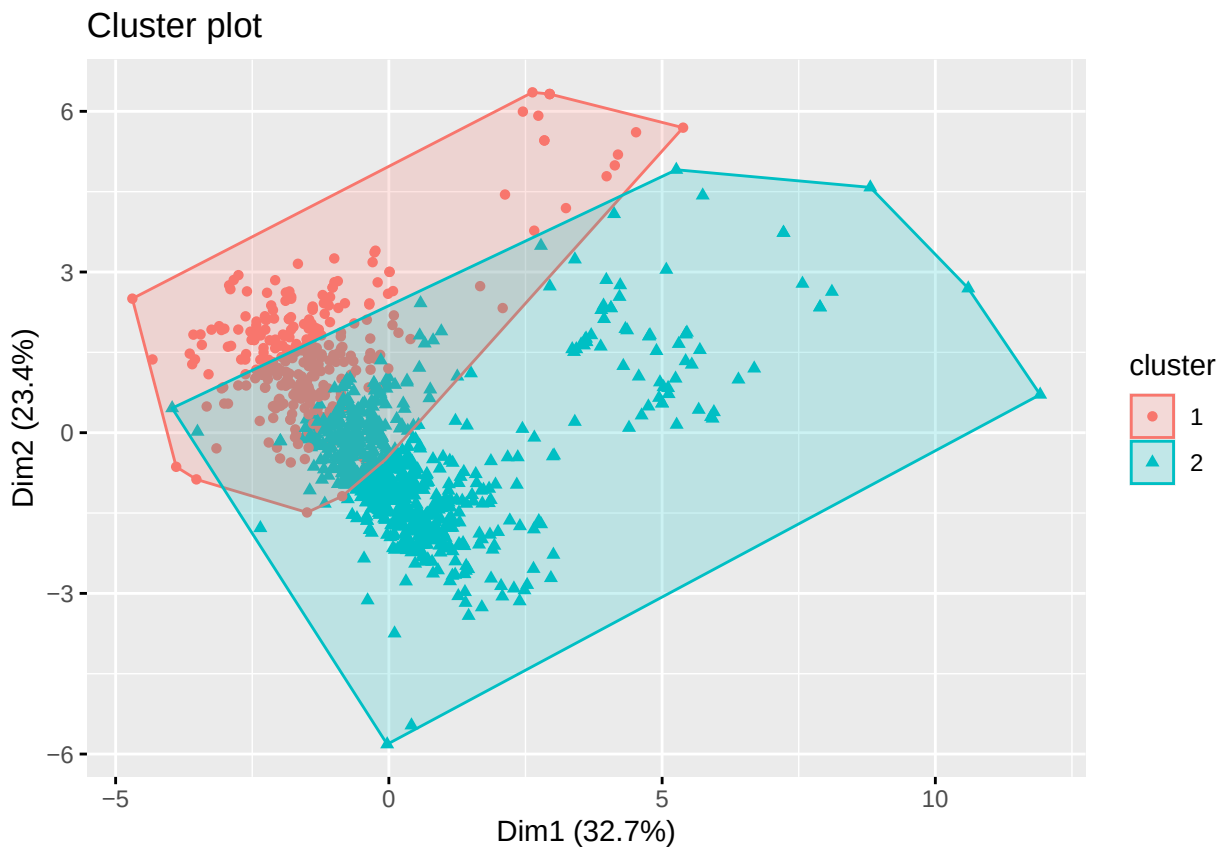
```
clara$medoids
```

```
##     Max.resolution Low.resolution Effective.pixels Zoom.wide Zoom.tele
## 898           1600            640                1        35        98
## 823           2560           2048                4        32        96
##     Normal.focus.range Macro.focus.range Storage.included Weight Dimensions
## 898                 50                10                8    340        106
## 823                 30                 1               12    190         91
##     Price
## 898   429
## 823   399
```

Nous remarquons que la première classe est composée de modèles d'appareils photos de faible résolutions (Max.resolution et Low.resolution moins élevées que dans la seconde classe), avec un poids et des dimensions élevés. Le prix est en moyenne plus élevé dans la première classe. La deuxième classe présente des modèles d'appareil photo avec des résolutions élevées, un poids et des dimensions faibles. Le prix est en moyenne plus faible que celui de la première classe.

Nous pouvons visualiser les différents clusters grâce à la fonction suivante :

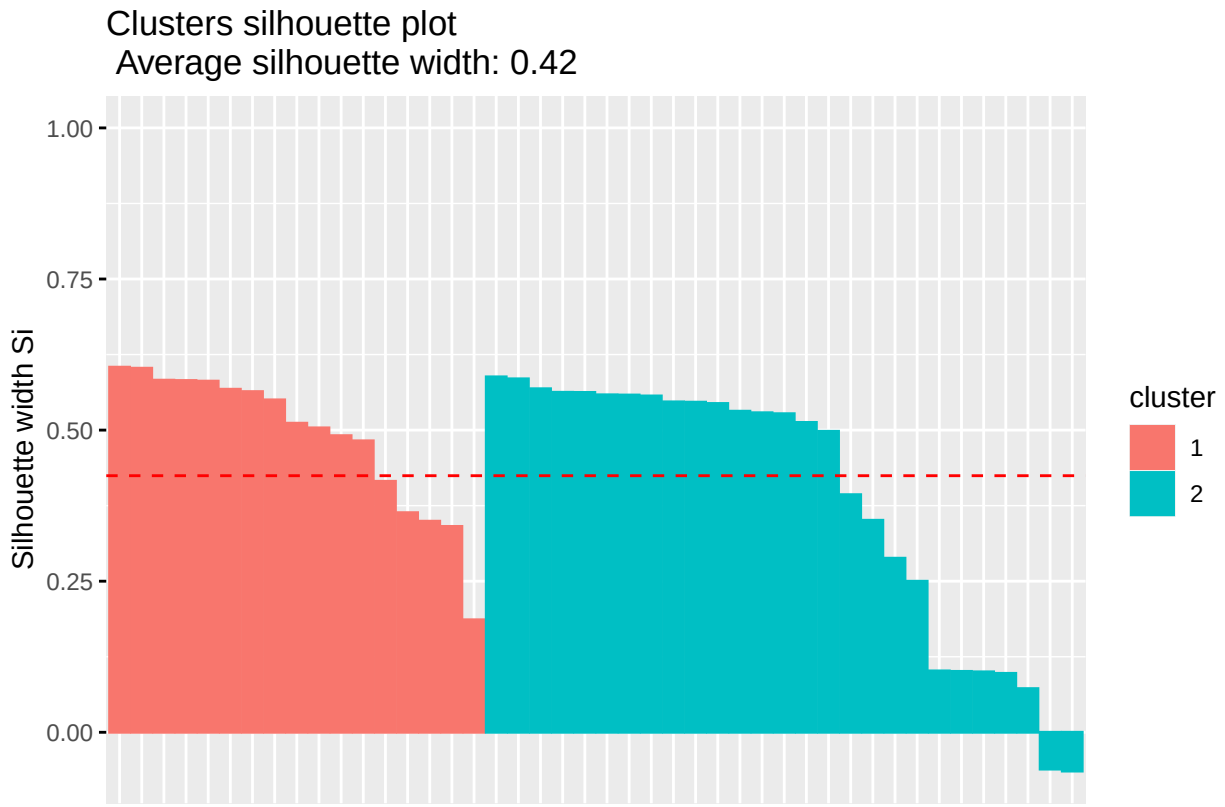
```
fviz_cluster (clara, data.quanti, geom="point")
```



Comme pour la méthode PAM, nous pouvons utiliser le coefficient de silhouette pour mesurer la qualité de la partition.

```
fviz_silhouette(silhouette(clara))
```

```
##   cluster size ave.sil.width
## 1       1   17          0.49
## 2       2   27          0.39
```



Nous remarquons, avec la méthode CLARA, qu'une classe contient des individus mal classés. Il s'agit de la classe 2, comme nous pouvons le constater sur le graphique précédent. En effet, nous observons une barre en dessous de la barre de 0, ce qui signifie que le coefficient de silhouette est négatif pour certains individus.

Une comparaison des différentes méthodes sera effectuée à la fin des différentes analyses.

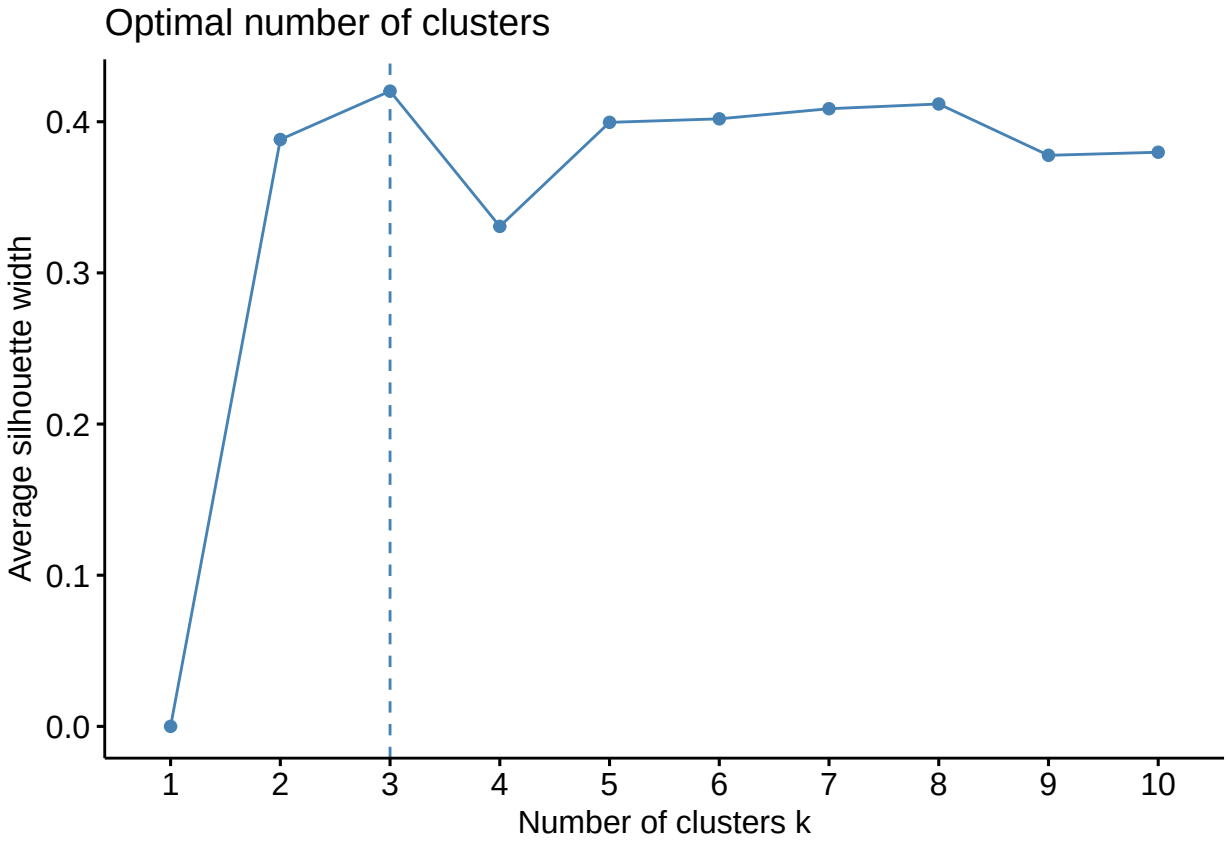
7. Classification mixte des données

Pour finir, nous allons réaliser une classification mixte des données. Pour cela, nous allons appliquer la méthode hclust de R.

```
CAH <- hclust(dist(data.quanti,method = 'euclidian'), method='ward.D2')
CAH
```

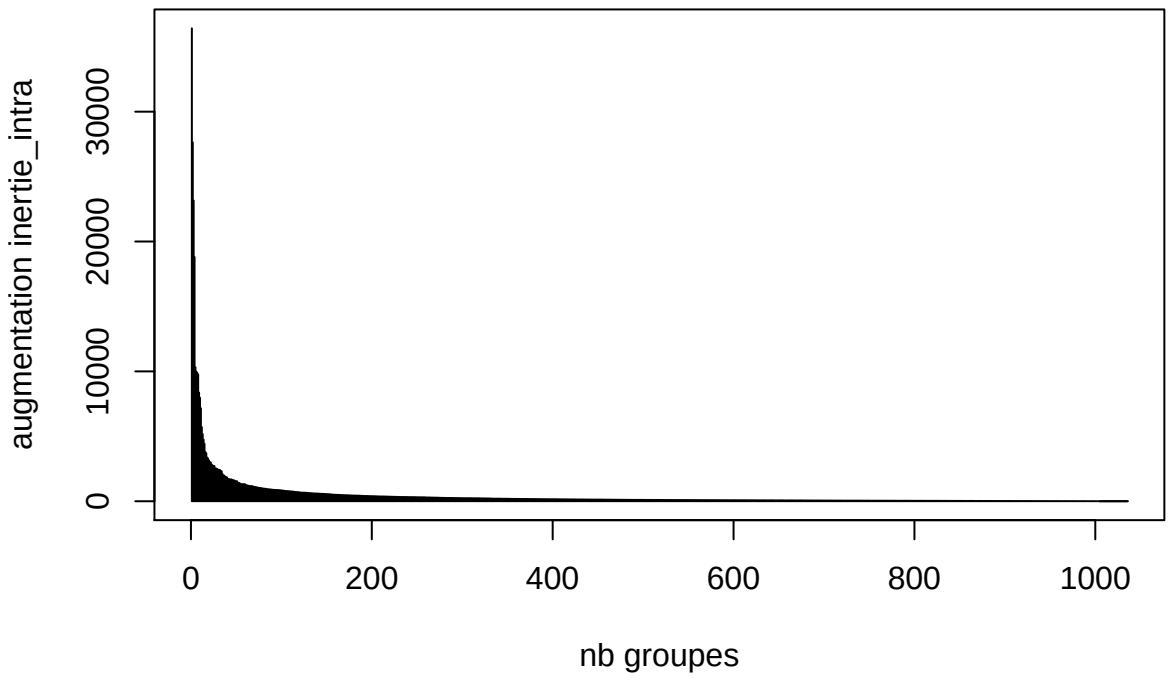
```
##
## Call:
## hclust(d = dist(data.quanti, method = "euclidian"), method = "ward.D2")
##
## Cluster method   : ward.D2
## Distance         : euclidean
## Number of objects: 1036
```

```
fviz_nbclust(data.quanti,factoextra::hcut,method="silhouette")
```



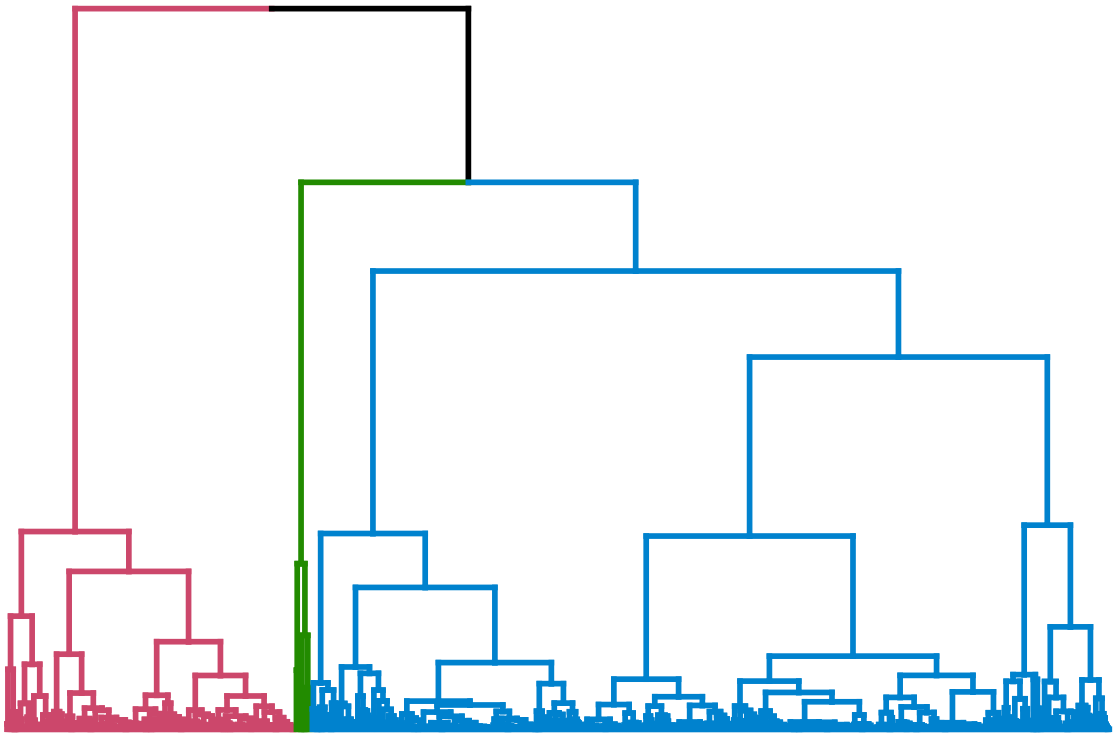
Grâce au graphique ci-dessus, nous pouvons décider de réaliser une segmentation en trois classes afin d'appliquer une classification mixte.

```
h <- CAH$height
plot(
  (nrow(data.quanti)-1):1,
  h,
  xlab="nb groupes",
  ylab="augmentation inertie_intra",
  type="h")
```



Nous pouvons afficher le dendogramme grâce à la fonction ci-dessous. Sur un axe apparait les individus à regrouper et sur l'autre sont indiqués les écarts correspondants aux différents niveaux de regroupement.

```
ggplot(color_branches(CAH, k = 3), labels = FALSE)
```



Pour obtenir le détail de la classification des groupes, nous pouvons utiliser la fonction cutree comme ci-dessous, qui va découper le jeu de données en trois classes :

```
groupes.cah <- cutree(CAH,k=3)

table(groupes.cah)
```

```
## groupes.cah
##   1   2   3
## 272 749  15
```

Nous obtenons ainsi le nombre d'individus présents dans chaque classe.

Nous pouvons calculer les différentes inerties avec la méthode CAH, qui permettront de mesurer la qualité de la méthode et ainsi de comparer les différentes méthodes effectuées.

```
groupes.cah <- cutree(CAH,k=3)
donnees_cah=cbind(data.quanti,groupes.cah)
setnames(donnees_cah, "groupes.cah", "cluster")

inertie_intra(donnees_cah,3)
```

```
## [1] 912120.5
```

```
inertie_totale(donnees_cah)
```

```
## [1] 1920775
```

```
inertie_inter(donnees_cah,3)
```

```
## [1] 1008654
```

```
inertie_expliquee(donnees_cah,3)
```

```
## [1] 52.51
```

En effectuant une classification mixte des données, nous obtenons un pourcentage d'inertie expliquée égal à 52,51%.

Nous voulons maintenant savoir si la classification mixte améliore les résultats. Pour cela, nous allons comparer les résultats avec les différentes méthodes utilisées précédemment. Nous allons utiliser les pourcentages d'inertie expliquée pour chaque méthode.

Pour comparer ces méthodes nous allons les appliquer sur un même nombre de classes à partitionner.

```
X=c()

k=3

dataLloyd=algo_Lloyd(k,data.quanti)
vect=c(
  "Méthode Lloyd implémentée",
  round(inertie_intra(dataLloyd,k),0),
  round(inertie_totale(dataLloyd),0),
  round(inertie_inter(dataLloyd,k),0),
  round(inertie_expliquee(dataLloyd,k),2))

X=rbind(X,vect)



kmeans.2 <- kmeans(data.quanti, centers=k, algorithm=c("Lloyd"))
inertie.expliquee=round((1-(kmeans.2$tot.withinss/kmeans.2$totss))*100,2)
vect=c(
  "Méthode Lloyd de R",
  round(kmeans.2$tot.withinss),
  round(kmeans.2$totss),
  round(kmeans.2$betweenss)
  ,round(inertie.expliquee))

X=rbind(X,vect)



pam = pam(data.quanti, k, metric = "euclidean")

donnees_pam=cbind(data.quanti,pam$clustering)
setnames(donnees_pam, "pam$clustering", "cluster")

vect=c(
  "Méthode PAM",
  round(inertie_intra(donnees_pam,k),0),
  round(inertie_totale(donnees_pam),0),
  round(inertie_inter(donnees_pam,k),0),
  round(inertie_expliquee(donnees_pam,k),2))

X=rbind(X,vect)
```

```r
clara = clara(data.quanti, k, metric = "euclidean") # samples par défaut : 5
donnees_clara=cbind(data.quanti,clara$clustering)
setnames(donnees_clara, "clara$clustering", "cluster")


vect=c(
  "Méthode Clara",
  round(inertie_intra(donnees_clara,k),0),
  round(inertie_totale(donnees_clara),0),
  round(inertie_inter(donnees_clara,k),0),
  round(inertie_expliquee(donnees_clara,k),2))

X=rbind(X,vect)


groupes.cah <- cutree(CAH,k)
donnees_cah=cbind(data.quanti,groupes.cah)
setnames(donnees_cah, "groupes.cah", "cluster")

vect=c(
  "Méthode CAH",
  round(inertie_intra(donnees_cah,k),0),
  round(inertie_totale(donnees_cah),0),
  round(inertie_inter(donnees_cah,k),0),
  round(inertie_expliquee(donnees_cah,k),2))

X=rbind(X,vect)

colnames(X)=c(
  "Méthode de classification",
  "Inertie Intra",
  "Inertie Totale",
  "Inertie Inter",
  "Inertie Expliquée")

kable(X)
```

	Méthode de classification	Inertie Intra	Inertie Totale	Inertie Inter	Inertie Expliquée
vect	Méthode Lloyd implémentée	105934	576116	470182	81.61
vect	Méthode Lloyd de R	996335952	1989922347	993586395	50
vect	Méthode PAM	911448	1920775	1009327	52.55
vect	Méthode Clara	908063	1920775	1012712	52.72
vect	Méthode CAH	912121	1920775	1008654	52.51

Nous constatons, en observant les différents pourcentages d'inertie expliquée, que la meilleure méthode est la méthode kMeans que nous avons implémenté avec un pourcentage d'inertie expliquée égal à 81,61%. Les méthodes PAM, CLARA et CAH sont quasiment similaires avec un pourcentage valant environ 52%. La méthode kMeans de R se classe en dernière position, avec un pourcentage égal à 50%. Nous pouvons donc affirmer que réaliser une classification mixte n'améliorera pas les résultats, comparée aux autres méthodes utilisées précédemment.

Conclusion

Ce projet nous a permis de mettre en oeuvre différentes méthodes et de les comparer entre elles. Après avoir effectué les différentes analyses, nous pouvons affirmer que la meilleure méthode pour traiter ce jeu de données est notre méthode implémentée de kMeans. En effet, celle-ci a un pourcentage d'inertie expliquée nettement plus élevé que toutes les autres méthodes. Celles-ci sont quasiment similaires puisque leur pourcentage d'inertie expliqué reste approximativement égal.

Bibliographie

- https://www.datanovia.com/en/lessons/clara-in-r-clustering-large-applications/
- https://www.datanovia.com/en/lessons/k-medoids-in-r-algorithm-and-practical-examples/
- http://eric.univ-lyon2.fr/~ricco/cours/slides/classif_centres_mobiles.pdf
- https://tel.archives-ouvertes.fr/tel-00195779/document
- https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-explo-classif.pdf
- https://math.univ-angers.fr/~labatte/enseignement%20UFR/master%20MIM/methodesnonsupervisee.pdf