Projet Analyse de données

BERBAGUI Amine - HACHANI Ghassen

22/05/2022

Contents

Ce jeu de données étudie les résultats en mathématiques de deux lycées au Portugal en fonction de plusieurs critères sociaux, démographiques et scolaires. On veut ainsi voir s'il y a une potentielle corrélation entre les notes d'un élève et les autres variables. Le but est d'appliquer les modèles de statistiques et de clustering vu tout au long de l'année. Pour plus d'informations sur la signification des variables, je vous invite à voir ce site directement :

http://archive.ics.uci.edu/ml/datasets/Student+Performance#

Importation des données

On importe notre csv de manière classique. On remarque déjà qu'il est assez volumineux : 33 colonnes et 395 lignes. On va nettoyer ce csv car certaines colonnes ne nous intéresse pas forcèment.

```
setwd("C:/Users/Amine/Documents/#Polytech/MAIN 4/S2/Analyse de données/Projet")
data = read.csv2("student/student-mat.csv",sep=";",header=TRUE)
head(data)
```

```
##   school sex age address famsize Pstatus Medu Fedu     Mjob     Fjob     reason
## 1     GP   F  18       U     GT3       A    4    4  at_home  teacher     course
## 2     GP   F  17       U     GT3       T    1    1  at_home    other     course
## 3     GP   F  15       U     LE3       T    1    1  at_home    other      other
```

```
## 4     GP   F  15       U     GT3       T    4    2   health services       home
## 5     GP   F  16       U     GT3       T    3    3    other    other       home
## 6     GP   M  16       U     LE3       T    4    3 services    other reputation
##   guardian traveltime studytime failures schoolsup famsup paid activities
## 1   mother          2         2        0       yes     no   no         no
## 2   father          1         2        0        no    yes   no         no
## 3   mother          1         2        3       yes     no  yes         no
## 4   mother          1         3        0        no    yes  yes        yes
## 5   father          1         2        0        no    yes  yes         no
## 6   mother          1         2        0        no    yes  yes        yes
##   nursery higher internet romantic famrel freetime goout Dalc Walc health
## 1     yes    yes       no       no      4        3     4    1    1      3
## 2      no    yes      yes       no      5        3     3    1    1      3
## 3     yes    yes      yes       no      4        3     2    2    3      3
## 4     yes    yes      yes      yes      3        2     2    1    1      5
## 5     yes    yes       no       no      4        3     2    1    2      5
## 6     yes    yes      yes       no      5        4     2    1    2      5
##   absences G1 G2 G3
## 1        6  5  6  6
## 2        4  5  5  6
## 3       10  7  8 10
## 4        2 15 14 15
## 5        4  6 10 10
## 6       10 15 15 15
```

Nettoyage des données

Dans un premier temps, on enlève les notes du S1(G1) , S2(G2) pour ne garder qu'une seule colonne représentant la moyenne finale car sinon, les colonnes G1,G2,G3 seront trop corréler entre elles. De plus, on a parfois des 0 comme moyennes qui peuvent poser prolème par la suite.

```
for(i in 1:dim(data)[1])
{
  data$Average[i] = (data$G1[i] + data$G2[i])/2
}
```

On peut maintenant supprimer les colonnes inutiles comme le nom de l'école, le sexe, l'adresse, etc...

```
# supression des colonnes inutiles
data$guardian = NULL
data$age      = NULL
data$nursery  = NULL
data$internet = NULL
data$reason   = NULL
data$address  = NULL
data$sex      = NULL
data$school   = NULL
data$G1       = NULL
data$Pstatus  = NULL
data$G2       = NULL
data$G3       = NULL
```

Comme l'on a plusieurs variables qualitatives binaire (oui/non), on les met en as factor directement pour nos futurs modèles.

```
size = dim(data)[2]
colName = names(data)
for(i in 1:size)
{
  col = data[i]

  if(dim(unique(col))[1] == 2)
  {
    data[i] = as.factor(unlist(data[i]))
  }
}
```

```
str(data)
```

```
## 'data.frame':    395 obs. of  22 variables:
##  $ famsize   : Factor w/ 2 levels "GT3","LE3": 1 1 2 1 1 2 2 1 2 1 ...
##  $ Medu      : int  4 1 1 4 3 4 2 4 3 3 ...
##  $ Fedu      : int  4 1 1 2 3 3 2 4 2 4 ...
##  $ Mjob      : chr  "at_home" "at_home" "at_home" "health" ...
##  $ Fjob      : chr  "teacher" "other" "other" "services" ...
##  $ traveltime: int  2 1 1 1 1 1 1 2 1 1 ...
##  $ studytime : int  2 2 2 3 2 2 2 2 2 2 ...
##  $ failures  : int  0 0 3 0 0 0 0 0 0 0 ...
##  $ schoolsup : Factor w/ 2 levels "no","yes": 2 1 2 1 1 1 1 2 1 1 ...
##  $ famsup    : Factor w/ 2 levels "no","yes": 1 2 1 2 2 2 1 2 2 2 ...
##  $ paid      : Factor w/ 2 levels "no","yes": 1 1 2 2 2 2 1 1 2 2 ...
##  $ activities: Factor w/ 2 levels "no","yes": 1 1 1 2 1 2 1 1 1 2 ...
##  $ higher    : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ romantic  : Factor w/ 2 levels "no","yes": 1 1 1 2 1 1 1 1 1 1 ...
##  $ famrel    : int  4 5 4 3 4 5 4 4 4 5 ...
##  $ freetime  : int  3 3 3 2 3 4 4 1 2 5 ...
##  $ goout     : int  4 3 2 2 2 2 4 4 2 1 ...
##  $ Dalc      : int  1 1 2 1 1 1 1 1 1 1 ...
##  $ Walc      : int  1 1 3 1 2 2 1 1 1 1 ...
##  $ health    : int  3 3 3 5 5 5 3 1 1 5 ...
##  $ absences  : int  6 4 10 2 4 10 0 6 0 0 ...
##  $ Average   : num  5.5 5 7.5 14.5 8 15 12 5.5 17 14.5 ...
```

Régression linéaire multiple

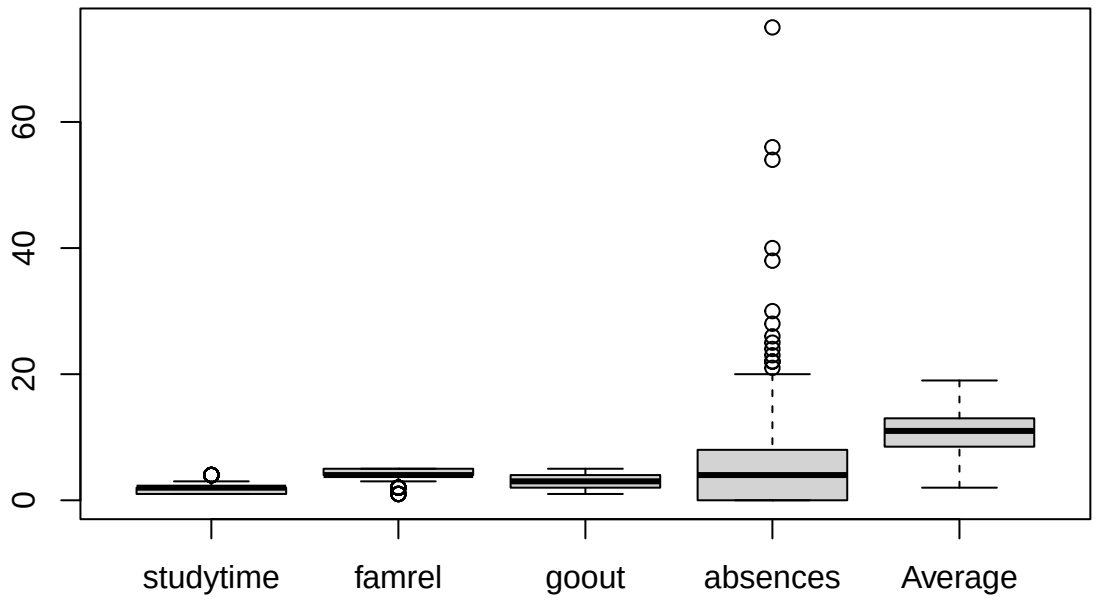
On décide d'appliquer un modèle de régression multiple en essayant d'expliquer **Average** en fonction de studytime (temps de travail) ,famrel (relation avec sa famille), goout (le temps de sorties) et absences. On a choisi ces variables car elles nous semblaient être les plus pertinents pour expliquer une réussite ou un échec scolaire.

```
dataRM = data
dataRM = data[,-c(1,2,3,4,5,6,8,9,10,11,12,13,14,16,18,19,20)]
str(dataRM)
```

```
## 'data.frame':    395 obs. of  5 variables:
##  $ studytime: int  2 2 2 3 2 2 2 2 2 2 ...
##  $ famrel   : int  4 5 4 3 4 5 4 4 4 5 ...
##  $ goout    : int  4 3 2 2 2 2 4 4 2 1 ...
##  $ absences : int  6 4 10 2 4 10 0 6 0 0 ...
##  $ Average  : num  5.5 5 7.5 14.5 8 15 12 5.5 17 14.5 ...
```

En traçant un boxplot de nos données, on s'aperçoit que les boites à moustaches sont très compréssées, ce qui est normal car la majorité de nos variables quantitatives prennent leurs valeurs seulement entre 0 et 5. La colonnes absences elle, contient des valeurs extrêmes comme on peut le voir. Cela correspond aux lycéens ayant un nombre d'absence très important

```
boxplot(dataRM)
```



On pose notre modèle linéaire

```
res=lm(Average ~., dataRM)
```

On vérifie la colinéarité entre nos variables. On a des valeurs $<$ à 10 donc c'est bon !

```
library(car)
```

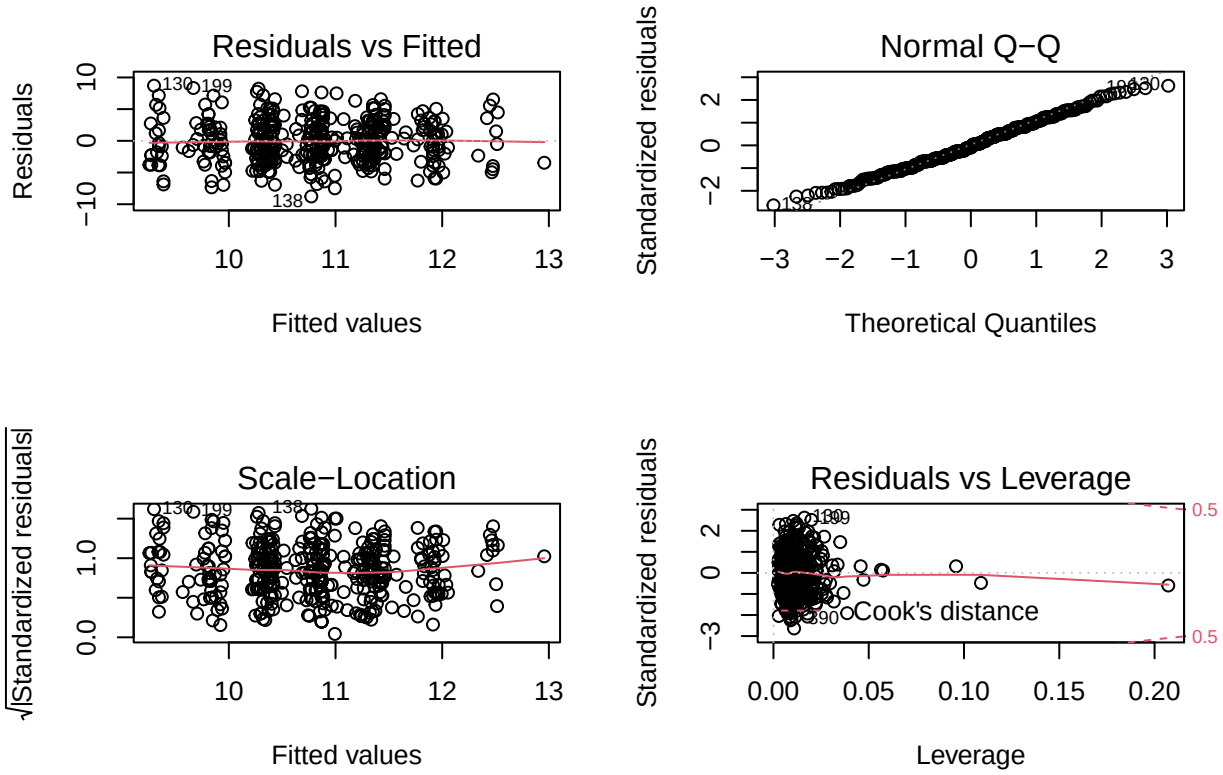
```
## Le chargement a nécessité le package : carData
```

```
vif(res)
```

```
## studytime    famrel     goout  absences
##  1.009458  1.008165  1.010574  1.007619
```

Analyse des résidus

```
par(mfrow=c(2,2))
plot(res)
```



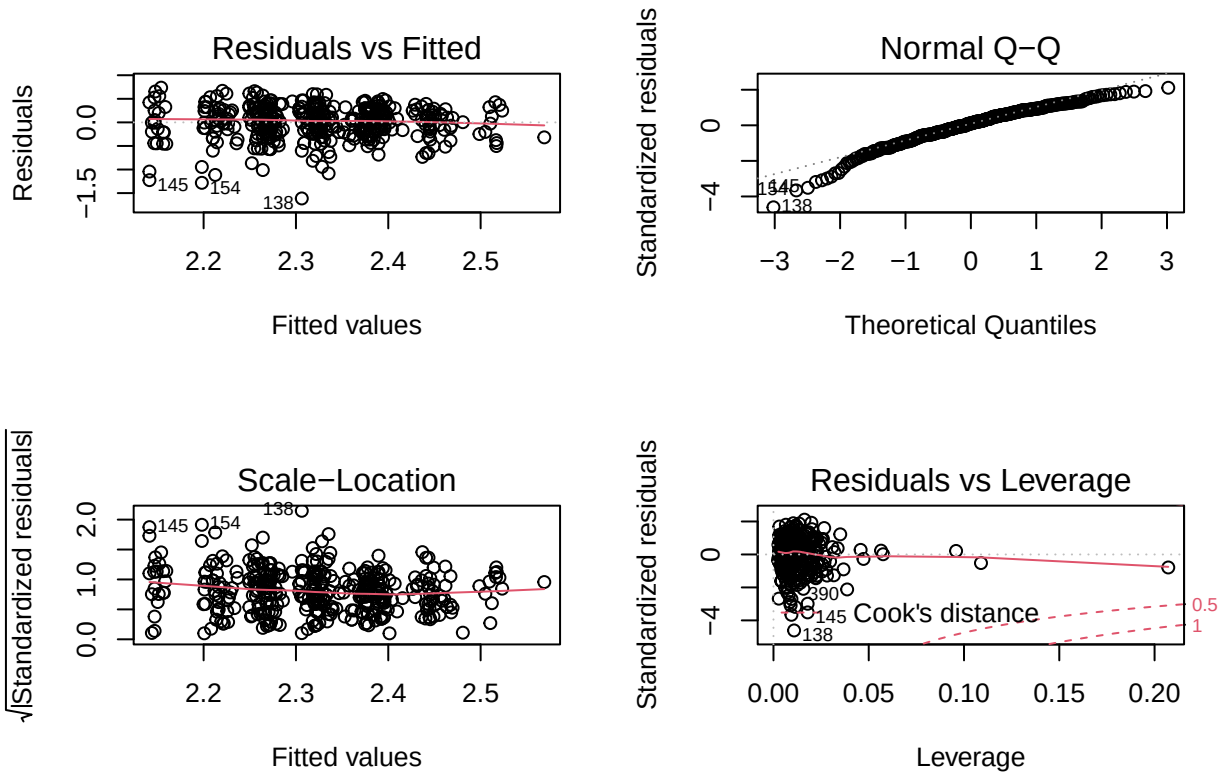
Au niveau de l'analyse des résidus, on remarque qu'il n'y a pas de "trompette" sur nos résidus qui vérifient l'homoscédasticité. En revanche, sur le dernier graphe en bas à droite, on voit que l'on a pas mal de valeurs en dehors de [-2,2] ce qui peut poser problème. Appliquons un test de Shapiro-Wilk pour voir la normalité.

```
shapiro.test(res$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res$residuals
## W = 0.99413, p-value = 0.1329
```

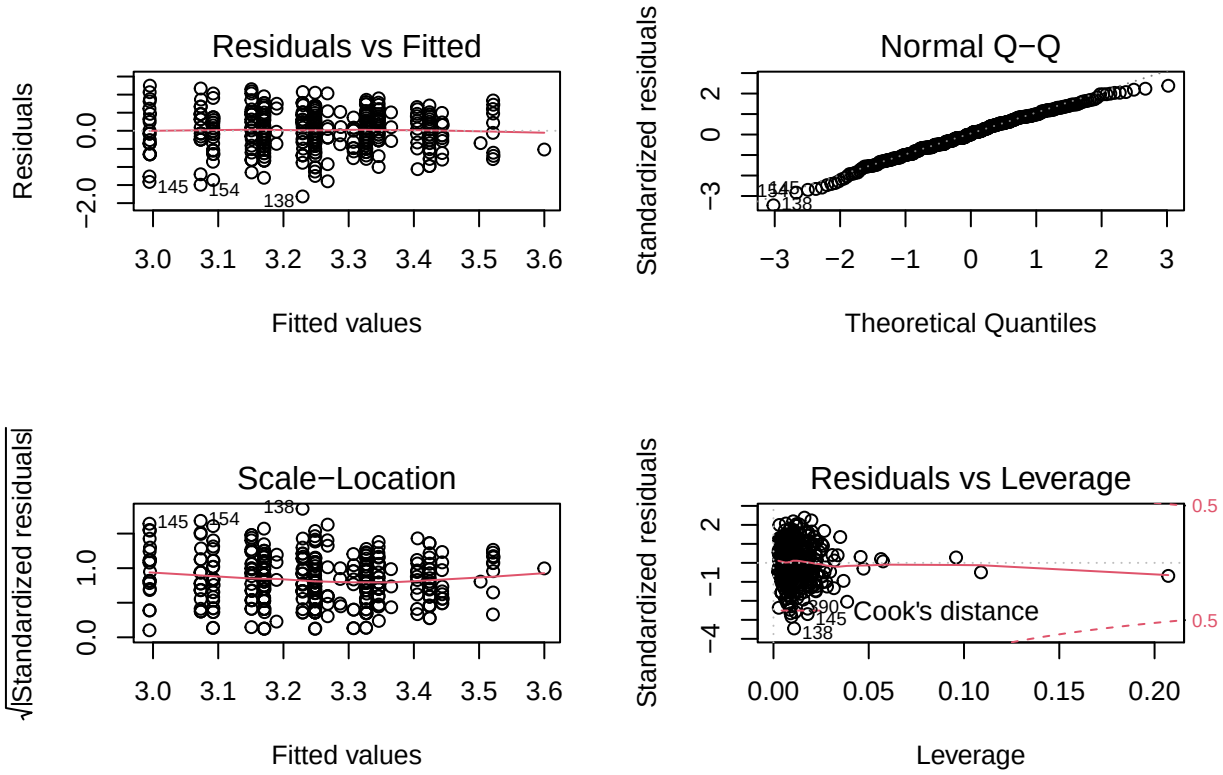
On a une p-value $= 0.13 > 5\%$ donc nos résidus sont bien gaussiens. Regardons maintenant ce que cela donne avec le log

```
res=lm(log(Average) ~., dataRM)
par(mfrow=c(2,2))
plot(res)
```



Le log ne semble pas arranger les choses, surtout au niveaux des valeurs atypiques. Voyons avec la racine carré.

```
res=lm(sqrt(Average) ~., dataRM)
par(mfrow=c(2,2))
plot(res)
```



C'est beaucoup mieux avec la fonction racine. Il reste quelques points extrême à enlever mais la majorité sont dans l'intervalle [-2,2]. En revanche lorsque l'on réapplique le test de Shapiro-Wilk, nos résidus ne sont plus gaussiens. Ainsi, nous allons garder le modèle standard sans fonction appliquée dessus.

```
shapiro.test(res$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res$residuals
## W = 0.99253, p-value = 0.04563
```

Suppression des points extrêmes

On récupère les points correspondants aux valeurs extrêmes

```
# on récupère les points plus grand que 2 en valeur absolue
abs(rstudent(res))[abs(rstudent(res))>2]
```

```
##       43       92      111      114      130      132      135      138
## 2.002646 2.041277 2.198767 2.048335 2.394406 2.197887 2.150213 3.502999
##      145      154      163      199      243      245      246      249
## 2.721004 2.862523 2.298207 2.246972 2.425819 2.676348 2.080944 2.380130
##      270      333      390
## 2.607944 2.485651 2.055205
```

On les enlève un à un à chaque fois

```
dataRM2=dataRM[-c(130),]
res=lm(Average ~., dataRM2)
par(mfrow=c(2,2))
#plot(res)
```

```
abs(rstudent(res))[abs(rstudent(res))>2]
```

```
##       43       92      105      111      114      138      145      154
## 2.321670 2.217823 2.002779 2.513583 2.396737 2.676125 2.077488 2.209300
##      199      245      246      249      266      270      307      333
## 2.581695 2.274294 2.367664 2.063226 2.203521 2.089858 2.192277 2.100625
##      375
## 2.281414
```

```
dataRM3=dataRM2[-c(130,138),]
res=lm(Average ~., dataRM3)
par(mfrow=c(2,2))
#plot(res)
```

La plupart de nos points extrêmes sont proche de 2 donc c'est bon.

```
abs(rstudent(res))[abs(rstudent(res))>2]
```

```
##       43       92      105      111      114      138      145      154
## 2.323959 2.237187 2.008974 2.522712 2.401474 2.686336 2.070126 2.207351
##      199      245      246      249      266      270      307      333
## 2.599930 2.284652 2.376763 2.071044 2.218194 2.088227 2.199564 2.104816
##      375
## 2.282312
```

```
shapiro.test(res$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res$residuals
## W = 0.99438, p-value = 0.1602
```

Points Leviers

Regardons si nos points de leviers risquent de perturber le modèle

```
dim(dataRM3)
```

```
## [1] 392   5
```

```
n = nrow(dataRM3)
p = ncol(dataRM3)
inflm.SR <- influence.measures(res)
leviers= inflm.SR$infmat[,"hat"]
2*p/n
```

```
## [1] 0.0255102
```

On a 23 observations avec des leviers trop grands. Voyons s'ils perturbent le modèle

```
leviers[leviers>2*p/n]
```

```
##         26         67         75        104        109        141        151
```

```
## 0.03689561 0.04585912 0.09597294 0.02551468 0.05667756 0.03184332 0.02855455
##        184        185        205        206        208        260        277
## 0.10904625 0.03374986 0.02761548 0.02680558 0.03201354 0.02793810 0.20757217
##        281        300        308        316        331        350        358
## 0.02986477 0.03517606 0.04744738 0.05760901 0.02570181 0.02643526 0.03105571
##        390        392
## 0.03873962 0.02707826
```

```
obs = 277
dataRM[obs,]
```

```
##     studytime famrel goout absences Average
## 277         2      4     1       75     9.5
```

On a visualisé les points leviers et ils ne semblent pas perturber le modèle. En revanche, on voit clairement que nos données ne représente pas un comportement linéaire nous permettant de faire de la régression. Comme nos variables quantitatives ne peuvent prendre qu'une valeur dans un intervalle réduit (0 à 5), cela nous donne des graphes avec des points homogénéiquement distribués en ligne ou en colonne mais ne formant pas de lien entre eux. Finalement, la régression multiplie ne semble pas être pertinente dans notre jeu de données.

Classification non supervisée

On passe maintenant à la classification non supervisée qui va nous permettre de faire du clustering sur nos données, c'est-à-dire de trouver une partition des individus en **K** classes à partir d'observations. On ne connait pas **K**, donc on fait une recherche à l'aveugle pour justement trouver un K optimal. On garde les variables qui nous intéresse

CAH

On applique la CAH pour trouver K. On crée un nouveau jeu de données correspondant. Comme la colonne **failures** est de niveau 4 (0,1,2,3), on l'a modifié de sorte à ce qu'elle soit de niveau 3, comme on avait en TP. Dans la boucle for, tous les étudiants qui ont redoublé plus de 2 fois passe à 2. Cela concerne un très faible échantillon

```
dataCAH = data
dataCAH = data[,-c(1,4,5,6,9,10,11,12,13,14,16,20,21)]
size = dim(data)[1]
for(i in 1:size)
{
  if(dataCAH$failures[i] > 2)
  {
    dataCAH$failures[i] = 2
  }
}
dataCAH$failures = as.factor(dataCAH$failures)
dataCAH <- dataCAH[c("failures", "Medu","Fedu", "studytime","goout", "Dalc","Walc","Average")]
str(dataCAH)
```

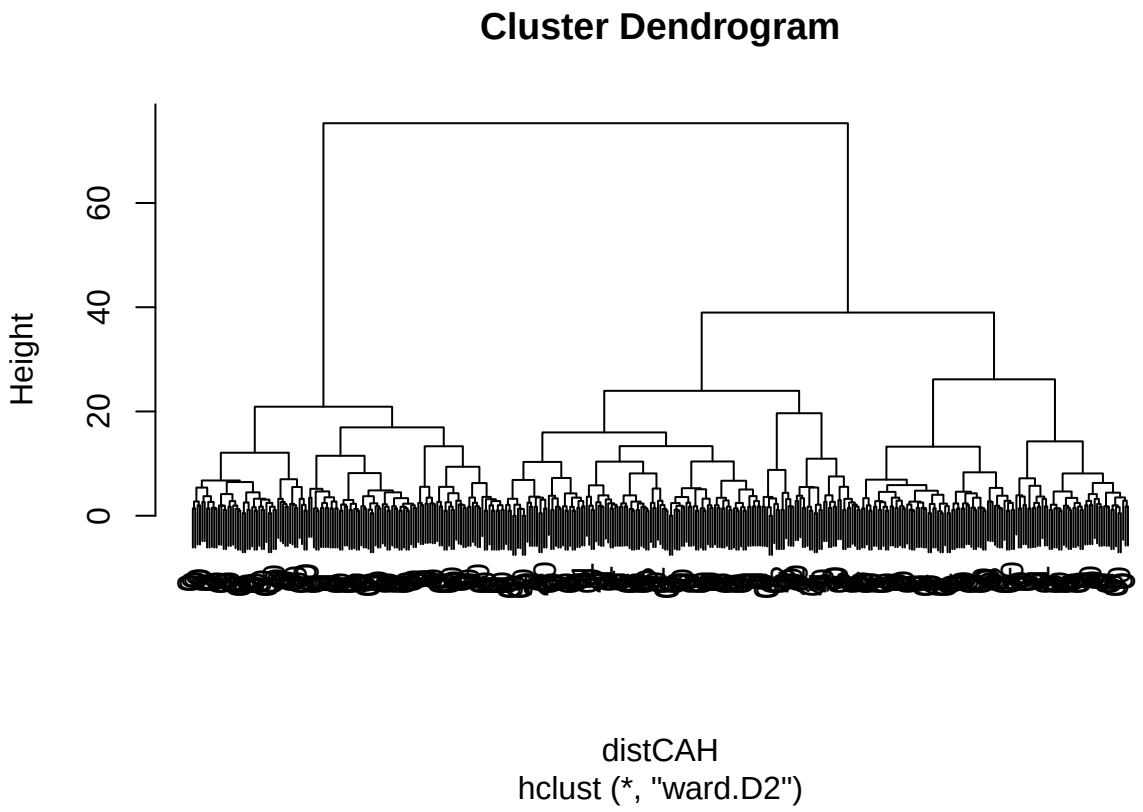
```
## 'data.frame':    395 obs. of  8 variables:
##  $ failures : Factor w/ 3 levels "0","1","2": 1 1 3 1 1 1 1 1 1 1 ...
##  $ Medu     : int  4 1 1 4 3 4 2 4 3 3 ...
##  $ Fedu     : int  4 1 1 2 3 3 2 4 2 4 ...
##  $ studytime: int  2 2 2 3 2 2 2 2 2 2 ...
##  $ goout    : int  4 3 2 2 2 2 4 4 2 1 ...
##  $ Dalc     : int  1 1 2 1 1 1 1 1 1 1 ...
##  $ Walc     : int  1 1 3 1 2 2 1 1 1 1 ...
##  $ Average  : num  5.5 5 7.5 14.5 8 15 12 5.5 17 14.5 ...
```

On calcule la matrice des distances entre nos individus avec la fonction *dist*, puis on construit le dendrogramme à l'aide de la mesure de Ward, qui est une méthode pour calculer les distances entre les classes.

```
distCAH = dist(dataCAH[,2:8])
hc <- hclust(distCAH, method = "ward.D2")
```

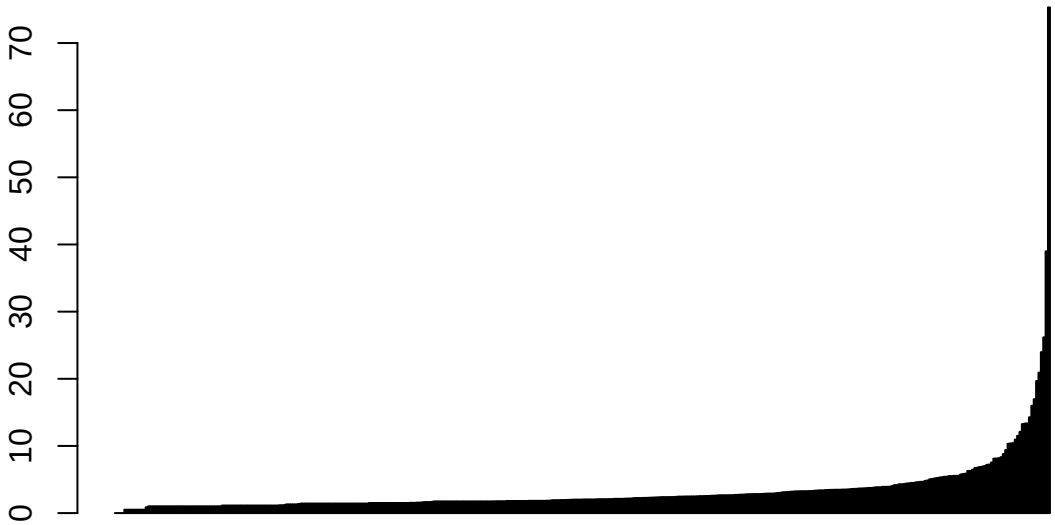
On plot ensuite notre dendrogramme. On voit qu'il contient beaucoup de sous classes mais que la démarcation se fait assez facilement.

```
plot(hc,labels=dataCAH$failures)
```



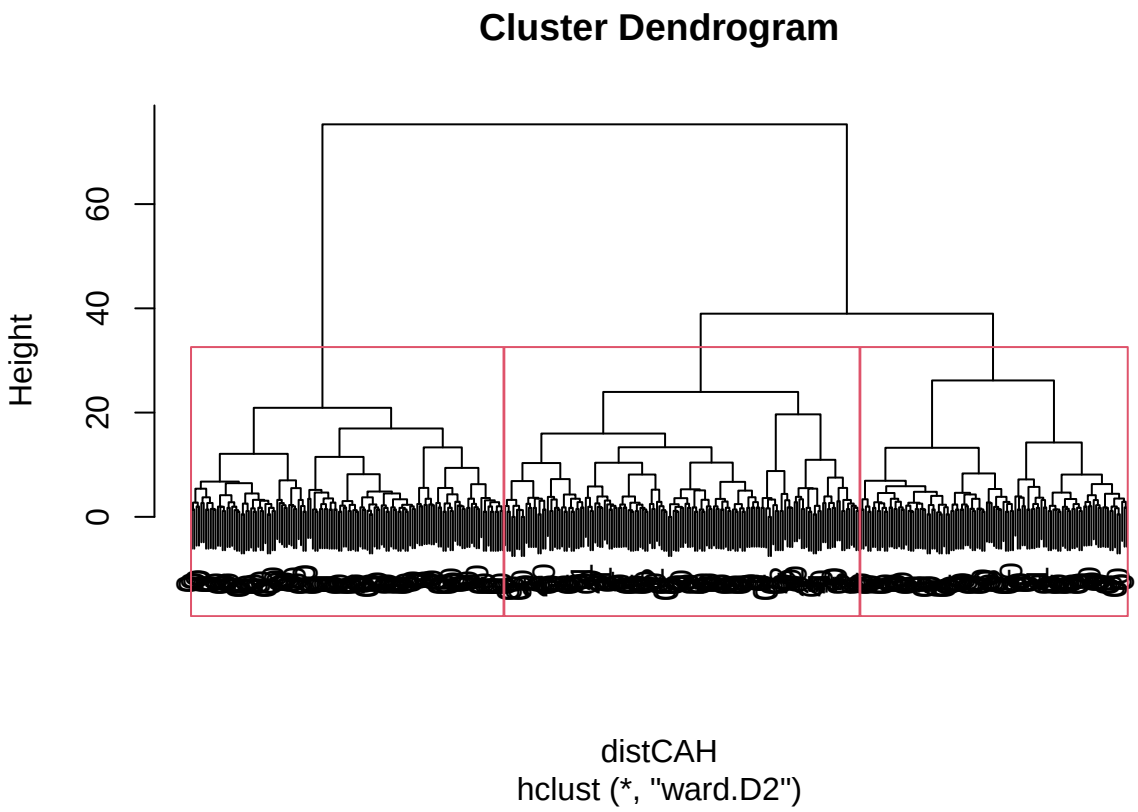
On peut aussi regarder le barplot, qui dans notre cas est assez compliqué à voir.

```
barplot(hc$height)
```



On utilise la fonction *rect.hclust* pour découper notre dendrogramme en **K** classes avec ici K = 3.

```
plot(hc,labels=dataCAH$failures)
rect.hclust(hc, k = 3)
```



On obtient les groupes:

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

```
##   [1] 1 1 1 2 1 2 3 1 2 2 1 3 2 3 2 2 2 1 1 3 2 2 2 2 3 1 3 2 3 3 3 2 2 1 2 1 2
##  [38] 2 2 3 1 2 2 1 1 1 2 2 2 1 2 2 3 3 3 1 2 2 1 2 3 1 1 3 3 2 3 1 1 2 2 1 1 3
##  [75] 2 3 1 3 1 1 3 3 1 2 1 3 1 2 3 1 1 2 1 3 3 1 2 1 2 1 1 2 3 1 2 1 1 2 3 2 2
## [112] 1 3 2 1 2 3 2 1 2 2 2 2 2 1 2 1 1 1 2 1 1 3 3 1 1 1 1 2 2 1 1 3 2 1 1 1 1
## [149] 1 1 1 3 3 1 3 1 2 1 2 3 1 1 1 1 1 3 3 2 1 2 1 3 2 1 1 3 2 1 3 3 3 2 2 3 3
## [186] 2 3 2 1 1 3 1 1 3 2 2 2 3 2 1 2 1 1 1 1 3 1 3 1 1 1 3 3 1 1 2 1 1 1 1 1 1
## [223] 2 3 2 1 2 3 1 3 2 3 3 2 1 3 3 3 3 1 2 3 1 2 1 2 3 1 1 2 1 1 1 1 1 2 3 2
## [260] 1 2 1 2 1 1 2 3 3 1 1 3 2 3 2 3 2 1 3 1 3 3 3 3 1 1 3 2 3 2 2 3 2 3 2 2 2
## [297] 3 1 2 2 3 3 2 2 2 2 2 2 1 2 3 1 3 3 3 3 3 1 1 3 3 2 1 3 2 2 3 2 3 3 2 3 2 1
## [334] 1 1 2 2 1 2 3 3 3 2 1 3 2 2 3 2 3 1 2 1 1 2 1 2 3 1 2 2 2 3 2 3 1 2 1 3 2
## [371] 1 2 3 1 2 1 2 3 2 3 2 1 3 1 1 3 1 1 1 1 3 2 1 3 1
```

```
table(groupes.cah)
```

```
## groupes.cah
##   1   2   3
## 150 132 113
```

On voit donc que l'on a 150 observations dans la classe 1, 132 dans la classe 2 et 113 dans la classe 3.

K-means

```
dataKmeans = dataCAH
str(dataKmeans)
```

```
## 'data.frame':    395 obs. of  8 variables:
##  $ failures : Factor w/ 3 levels "0","1","2": 1 1 3 1 1 1 1 1 1 1 ...
##  $ Medu     : int  4 1 1 4 3 4 2 4 3 3 ...
##  $ Fedu     : int  4 1 1 2 3 3 2 4 2 4 ...
##  $ studytime: int  2 2 2 3 2 2 2 2 2 2 ...
##  $ goout    : int  4 3 2 2 2 2 4 4 2 1 ...
##  $ Dalc     : int  1 1 2 1 1 1 1 1 1 1 ...
##  $ Walc     : int  1 1 3 1 2 2 1 1 1 1 ...
##  $ Average  : num  5.5 5 7.5 14.5 8 15 12 5.5 17 14.5 ...
```

Maintenant qu'on connait K, on peut le passer en entrée dans K-means

```
K = 3
kmeans.result=kmeans(dataKmeans[,2:8], centers = K)
names(kmeans.result)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
kmeans.result$size
```

```
## [1] 163 107 125
```

```
kmeans.result$centers
```

```
##       Medu     Fedu studytime    goout     Dalc     Walc  Average
## 1 2.803681 2.650307  2.079755 3.110429 1.607362 2.435583 10.92638
## 2 3.140187 2.803738  2.196262 2.859813 1.205607 1.887850 15.07009
## 3 2.344000 2.112000  1.840000 3.320000 1.552000 2.448000  7.01600
```

```
#kmeans.result$cluster


kmeans.result$totss  # Inertie totale
```

Inerties intra et inter classes

```
## [1] 7245.403
```

```
kmeans.result$withinss # Inertie intra-classes
```

```
## [1] 1318.8221  847.9065 1210.4320
```

```
kmeans.result$betweenss #Inertie inter-classes
```

```
## [1] 3868.242
```

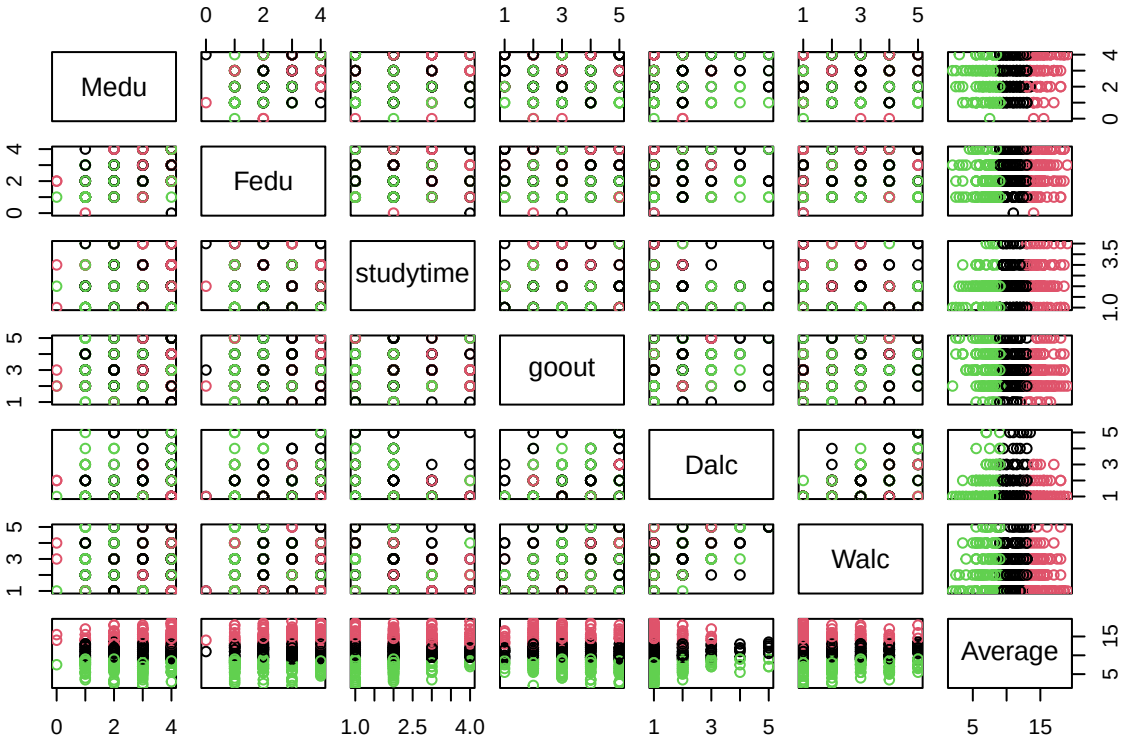
On peut comparer la classification obtenue avec la repartition en 3 classes :

```
table(dataCAH$failures, kmeans.result$cluster)
```

```
##
##       1   2   3
##   0 142 100  70
##   1  19   5  26
##   2   2   2  29
```

En regardant la sortie de table, on remarque que l'algorithme des K-means a vraiment eu du mal à classer les étudiants dans les 3 classes. En traçant la représentation des variables 2 à 2, on voit facilement qu'il y a très peu voire aucune linéarité entre les variables donc aucune corrélation ce qui fait que l'algorithme a du mal à partitionner efficacement **n** individus en **K** classes d'observations

```
pairs(dataCAH[,2:8], col=kmeans.result$cluster)
```



On se sert de la fonction adjustedRandIndex de la libraire mclust pour calculer l'ARI entre nos deux classifications et on a un score de 0.52 donc très moyen

```
library("mclust")
```

```
## Package 'mclust' version 5.4.9
## Type 'citation("mclust")' for citing this R package in publications.
```

```
adjustedRandIndex(kmeans.result$cluster,groupes.cah)
```

```
## [1] 0.5223843
```

Classification supervisée

Maintenant que l'on connait le nombre de cluster, nous pouvons passé à la classification supervisée pour faire de la prédiction mais aussi pour pouvoir trouver une combinaison de variables nous permettant de séparer au mieux les n individus en K groupes.

ACP

On applique une méthode d'analyse des composantes principales qui consiste à projetter au mieux nos variables dans un plan en maximisant l'inertie totale.

```
dataPCA = data[,-c(4,5,6,9,10,11,12,13,14,15,16,20,21)]
str(dataPCA)
```

```
## 'data.frame':    395 obs. of  9 variables:
##  $ famsize  : Factor w/ 2 levels "GT3","LE3": 1 1 2 1 1 2 2 1 2 1 ...
##  $ Medu     : int  4 1 1 4 3 4 2 4 3 3 ...
##  $ Fedu     : int  4 1 1 2 3 3 2 4 2 4 ...
##  $ studytime: int  2 2 2 3 2 2 2 2 2 2 ...
##  $ failures : int  0 0 3 0 0 0 0 0 0 0 ...
##  $ goout    : int  4 3 2 2 2 2 4 4 2 1 ...
##  $ Dalc     : int  1 1 2 1 1 1 1 1 1 1 ...
##  $ Walc     : int  1 1 3 1 2 2 1 1 1 1 ...
##  $ Average  : num  5.5 5 7.5 14.5 8 15 12 5.5 17 14.5 ...
```

On applique une ACP centrée réduite sur nos données

```
library(FactoMineR)
res = PCA(dataPCA,scale.unit = TRUE, graph = FALSE, quali.sup = 1)
```

Valeurs propres et parts d'inertie expliquée par chaque axe

```
res$eig
```

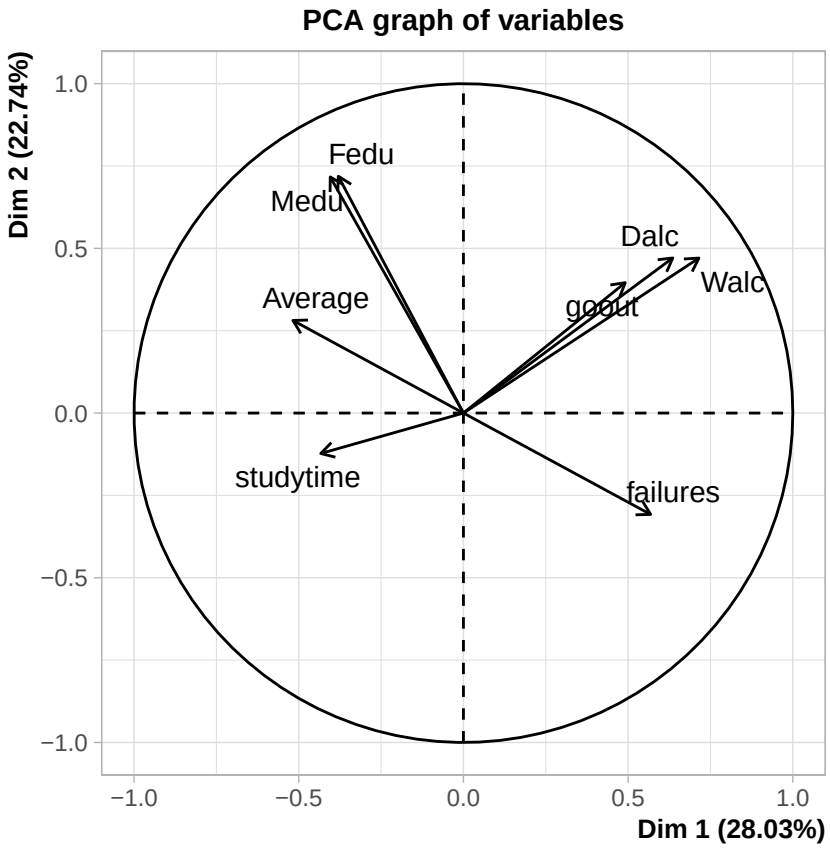
```
##        eigenvalue percentage of variance cumulative percentage of variance
## comp 1  2.2425569              28.031961                           28.03196
## comp 2  1.8192002              22.740003                           50.77196
## comp 3  1.0191134              12.738918                           63.51088
## comp 4  0.9302555              11.628194                           75.13908
## comp 5  0.6856998               8.571247                           83.71032
## comp 6  0.6152255               7.690319                           91.40064
## comp 7  0.3785340               4.731675                           96.13232
## comp 8  0.3094146               3.867683                          100.00000
```

Barplot associé. On remarque que les deux premières composantes suffisent à expliquer 50% de la variance totale

```
#barplot(res$eig[,2],col="blue")
```

Graphe des variables

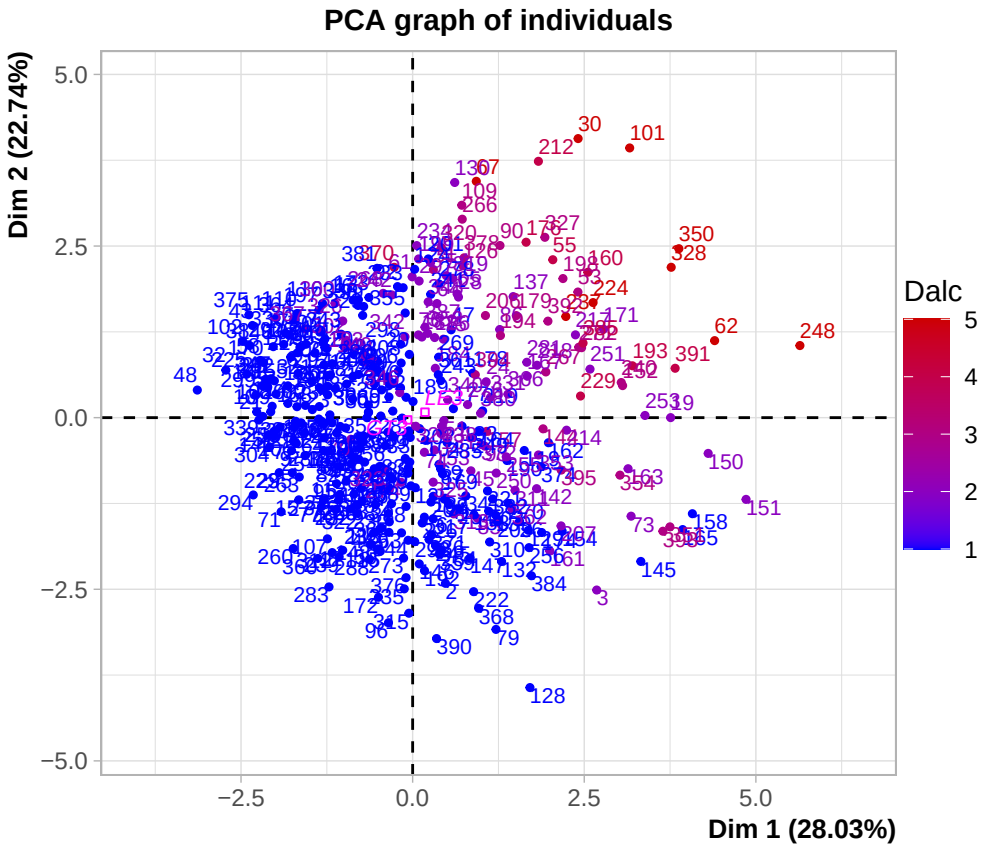
```
plot(res,choix = "var")
```



On remarque qu'il y a des variables pas très bien projetés comme **studytime** ou **Average**.Les deux axes expliquent à peine 50 % de l'inertie totale ce qui est très faible par rapport à ce que l'on avait en TP. Sur l'axe 1, on peut voir à droite les variables **Dalc**, **Walc** et **goout** corrélés positivement tandis que **studytime** est corrélé négativement. On a donc à droite les étudiants ne travaillant pas beaucoup, sortant assez souvent pour boire, etc... tandis qu'à gauche ceux qui travaillent plus et qui ont plus de chance d'avoir une moyenne haute, ce qui semble logique. Sur l'axe 2, c'est un peu plus compliqué à distinguer mais on voit que **Medu** et **Fedu** sont bien projetés et corrélés positivement. Ainsi, on peut supposer que l'axe 2 oppose les étudiants dont les parents ont fais des études (en haut) contre ceux n'ayant pas fais d'études (en bas)

Graphe des individus : on voit clairement deux groupes : en bleu, les etudiants avec une bonne moyenne, sérieux et en rouge les étudiants festifs et travaillant moins. Toutefois, les points sont assez mélangés.

```
plot(res,choix = "ind", habillage = 7, cex = 0.7)
```



Nous allons appliquer plusieurs méthode d'analyse discriminante que nous avons vu, puis effectuer une prédiction avec chacun d'eux.

LDA

```
library(MASS)
dataLDA = dataCAH
str(dataLDA)
```

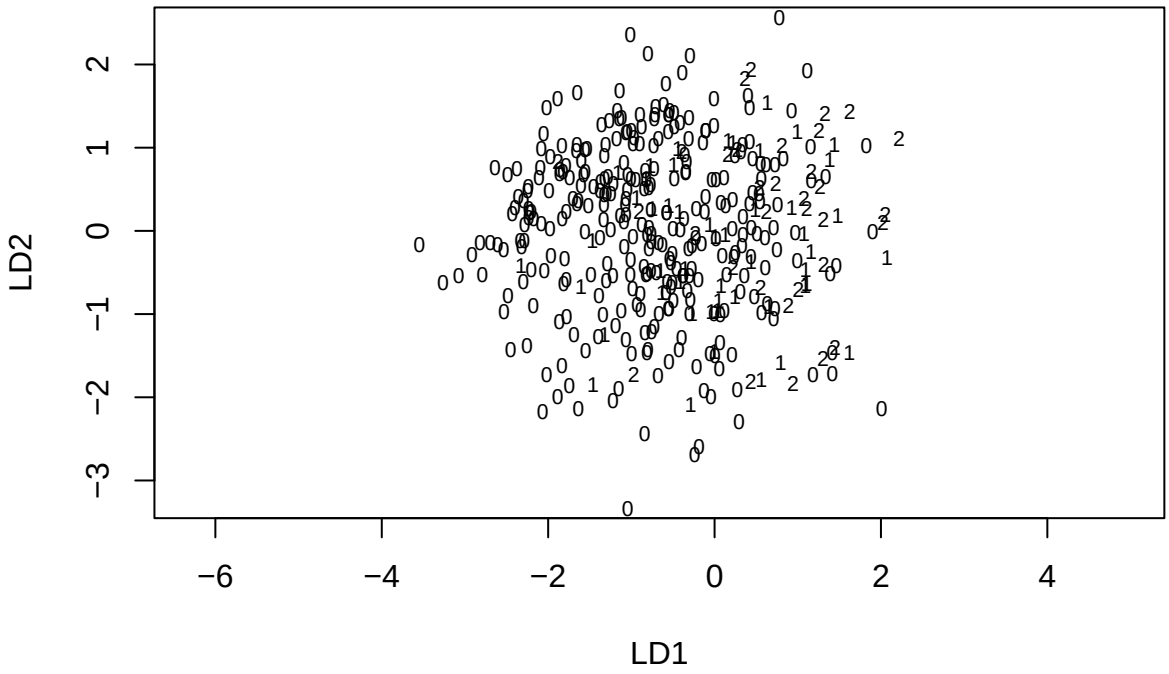
```
## 'data.frame':    395 obs. of  8 variables:
##  $ failures : Factor w/ 3 levels "0","1","2": 1 1 3 1 1 1 1 1 1 1 ...
##  $ Medu     : int  4 1 1 4 3 4 2 4 3 3 ...
##  $ Fedu     : int  4 1 1 2 3 3 2 4 2 4 ...
##  $ studytime: int  2 2 2 3 2 2 2 2 2 2 ...
##  $ goout    : int  4 3 2 2 2 2 4 4 2 1 ...
##  $ Dalc     : int  1 1 2 1 1 1 1 1 1 1 ...
##  $ Walc     : int  1 1 3 1 2 2 1 1 1 1 ...
##  $ Average  : num  5.5 5 7.5 14.5 8 15 12 5.5 17 14.5 ...
```

```
res.afd.lda = lda(failures ~., data=dataLDA)
res.afd.lda
```

```
## Call:
## lda(failures ~ ., data = dataLDA)
##
## Prior probabilities of groups:
##         0         1         2
```

```
## 0.7898734 0.1265823 0.0835443
##
## Group means:
##       Medu     Fedu studytime    goout     Dalc     Walc   Average
## 0 2.884615 2.647436  2.102564 3.051282 1.416667 2.205128 11.408654
## 1 2.260000 2.160000  1.860000 3.240000 1.680000 2.500000  9.410000
## 2 2.212121 1.878788  1.666667 3.454545 1.787879 2.787879  7.287879
##
## Coefficients of linear discriminants:
##                     LD1         LD2
## Medu      -0.198408484  1.03190529
## Fedu      -0.295530098 -0.41659212
## studytime -0.281858879 -0.01566475
## goout      0.093352928 -0.05946450
## Dalc       0.252965190 -0.58613368
## Walc       0.005514422  0.34423540
## Average   -0.220978194 -0.13894894
##
## Proportion of trace:
##    LD1    LD2
## 0.9653 0.0347
```

```
plot(res.afd.lda)
```



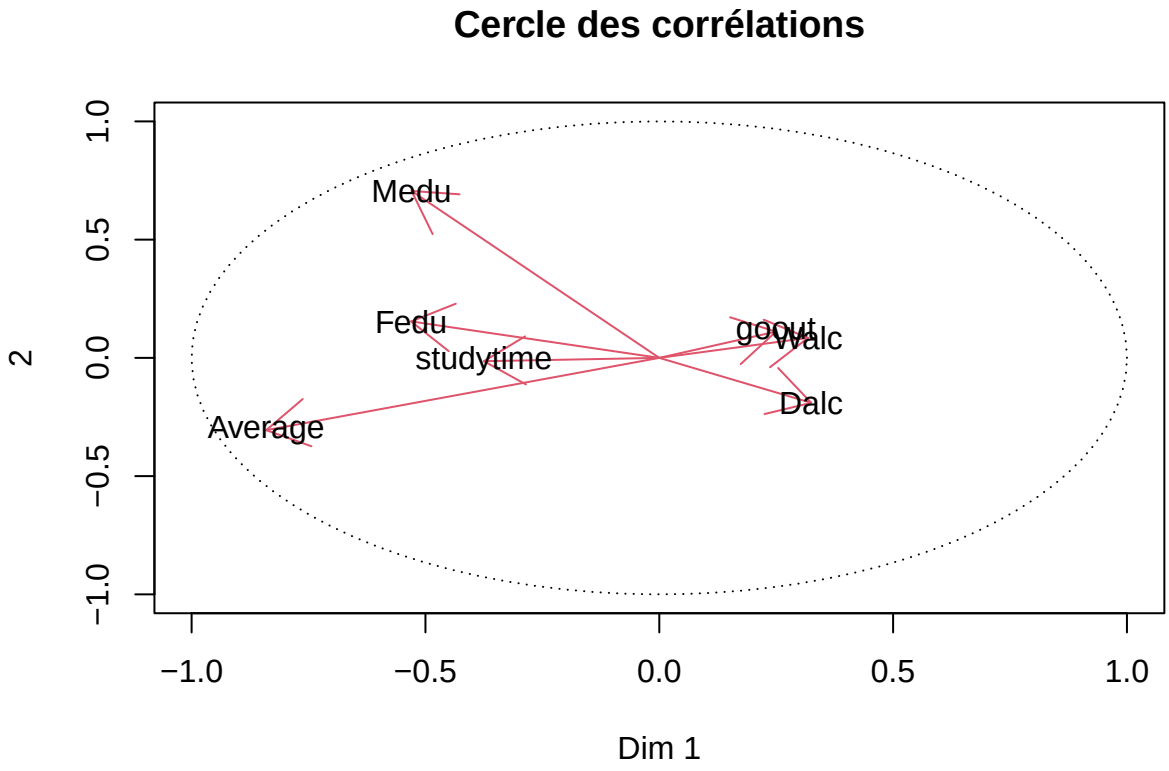
Les points sont mélangés et il est assez difficile d'en tirer des informations.

Traçcage du cercle des corrélations

```
F12 = predict(res.afd.lda, prior=rep(1/3,3))$x
cercle_correlation=cor(dataLDA[,2:8],F12)
cercle_correlation
```

```
##                  LD1         LD2
## Medu      -0.5296234  0.70582775
## Fedu      -0.5305909  0.15415406
## studytime -0.3749606 -0.01408700
## goout      0.2494269  0.11107950
## Dalc       0.3248591 -0.18999817
## Walc       0.3180534  0.08440944
## Average   -0.8404922 -0.30574598
```

```
a=seq(0,2* pi,length=100)
plot(cos(a), sin(a), type='l',lty=3,xlab='Dim 1', ylab='Dim
2',main="Cercle des corrélations" )
arrows(0,0,cercle_correlation[,1],cercle_correlation[,2],col=2)
text(cercle_correlation,labels=colnames(dataLDA[,2:8]))
```



On retrouve ce que l'on avait dans le graphe des variables. Les variables Dalc,Walc,goout sont très mal projetés, tandis que Average, Medu, Fedu eux sont bien projetés. La signification des axes est la même que pour le graphe des variables.

QDA

```
res.qda = qda(failures ~., data=dataLDA)
res.qda
```

```
## Call:
## qda(failures ~ ., data = dataLDA)
##
## Prior probabilities of groups:
##         0         1         2
## 0.7898734 0.1265823 0.0835443
##
## Group means:
##       Medu     Fedu studytime    goout     Dalc     Walc   Average
## 0 2.884615 2.647436  2.102564 3.051282 1.416667 2.205128 11.408654
## 1 2.260000 2.160000  1.860000 3.240000 1.680000 2.500000  9.410000
## 2 2.212121 1.878788  1.666667 3.454545 1.787879 2.787879  7.287879
```

Prédictions avec des nouveaux individus

On veut à présent savoir si ces modèles prédisent bien un nouvel échantillon dans notre jeu de données, c'est-à-dire un nouvel étudiant. Pour comparer, on a crée deux dataframe, un représentant un élève sérieux avec de bons résultats et un autre moins sérieux.

```
newdataGood= data.frame(Medu = 5, Fedu = 5, studytime = 4, goout = 1, Dalc = 0, Walc = 1,Average = 17.0)
newdataBad= data.frame(Medu = 1, Fedu = 1, studytime = 0, goout = 4, Dalc = 4, Walc = 4, Average = 6.5)
newdataGood
```

```
##   Medu Fedu studytime goout Dalc Walc Average
## 1    5    5         4     1    0    1      17
```

```
newdataBad
```

```
##   Medu Fedu studytime goout Dalc Walc Average
## 1    1    1         0     4    4    4     6.5
```

Avec LDA:

```
pred.ldaGood = predict(res.afd.lda,newdataGood)
pred.ldaBad = predict(res.afd.lda,newdataBad)
pred.ldaGood$posterior
```

```
##           0           1           2
## 1 0.9953614 0.004443433 0.0001951979
```

```
pred.ldaBad$posterior
```

```
##           0         1         2
## 1 0.1341036 0.3461221 0.5197743
```

Le bon élève a été prédit qu'il n'a jamais redoublé avec une probabilité de 99.6 %. Le mauvais élève lui a été prédit qu'il a redoublé 2 fois avec une probabilité de 51.9%.Nos prédictions sont plutôt cohérentes même si elles sont à prendre avec précaution. **Avec QDA**:

```
pred.qdaGood = predict(res.qda,newdataGood)
pred.qdaBad = predict(res.qda,newdataBad)
pred.qdaGood$posterior
```

```
##           0           1           2
## 1 0.9998385 0.0001265318 3.498325e-05
```

```
pred.qdaBad$posterior
```

```
##           0         1         2
## 1 0.1778878 0.3422088 0.4799035
```

Le bon élève a été prédit qu'il n'a jamais redoublé avec une probabilité de 99.9 %. Le mauvais élève lui a été prédit qu'il a redoublé 2 fois avec une probabilité de 48%. L'analyse quadratique semble plus efficace dans la prédiction que l'analyse linéaire.

Prédiction avec jeu de données/jeu de tests et courbe ROC

On va diviser notre jeu de données de manière à avoir 80% d'entre eux qui servent à l'entraînement et 20% aux tests.

```
set.seed(1)
n <- nrow(dataLDA)
p <- ncol(dataLDA)-1
test.ratio <- .2 # ratio of test/train samples
n.test <- round(n*test.ratio)
n.test
```

```
## [1] 79
```

```
tr <- sample(1:n,n.test)
data.test <- dataLDA[tr,]
data.train <- dataLDA[-tr,]
```

On applique **QDA** et **LDA**, puis on comparera nos résultats à l'aide de la courbe **ROC**

```
#QDA
res.afd.qda = qda(failures ~., data=data.train)
res.afd.qda
```

```
## Call:
## qda(failures ~ ., data = data.train)
##
## Prior probabilities of groups:
##          0          1          2
## 0.78797468 0.12658228 0.08544304
##
## Group means:
##       Medu     Fedu studytime    goout     Dalc     Walc   Average
## 0 2.871486 2.626506  2.084337 3.052209 1.421687 2.216867 11.287149
## 1 2.125000 2.050000  1.850000 3.250000 1.475000 2.450000  9.312500
## 2 2.185185 1.962963  1.740741 3.333333 1.777778 2.703704  7.592593
```

```
#LDA
res.afd.lda = lda(failures ~., data=data.train)
res.afd.lda
```

```
## Call:
## lda(failures ~ ., data = data.train)
##
## Prior probabilities of groups:
##          0          1          2
## 0.78797468 0.12658228 0.08544304
##
## Group means:
##       Medu     Fedu studytime    goout     Dalc     Walc   Average
## 0 2.871486 2.626506  2.084337 3.052209 1.421687 2.216867 11.287149
## 1 2.125000 2.050000  1.850000 3.250000 1.475000 2.450000  9.312500
## 2 2.185185 1.962963  1.740741 3.333333 1.777778 2.703704  7.592593
##
```

```
## Coefficients of linear discriminants:
##                   LD1         LD2
## Medu      -0.27732078  0.73879876
## Fedu      -0.28444705 -0.07124924
## studytime -0.22827945  0.13844660
## goout      0.07370219 -0.24174587
## Dalc       0.14885426  0.78986864
## Walc       0.05629970 -0.12161780
## Average   -0.21878708 -0.15765205
##
## Proportion of trace:
##    LD1    LD2
## 0.9401 0.0599
```

On effectue maintenant les prédicitons avec la fonction *predict*

```
#LDA
pred.lda = predict(res.afd.lda,data.test)
#pred.lda
#QDA
pred.qda = predict(res.afd.qda,data.test)
#pred.qda
```

On trace maintenant les courbes ROC.

```
# Aire sous la courbe
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attachement du package : 'pROC'
```

```
## Les objets suivants sont masqués depuis 'package:stats':
##
##     cov, smooth, var
```

```
#proba a posteriori de succes (dans la deuxi?me colonne) :
pred_lda <- pred.lda$posterior[,2]
pred_qda <- pred.qda$posterior[,2]
```

```
ROC_lda <- roc(data.test$failures, pred_lda)
```

```
## Warning in roc.default(data.test$failures, pred_lda): 'response' has more than
## two levels. Consider setting 'levels' explicitly or using 'multiclass.roc'
## instead
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

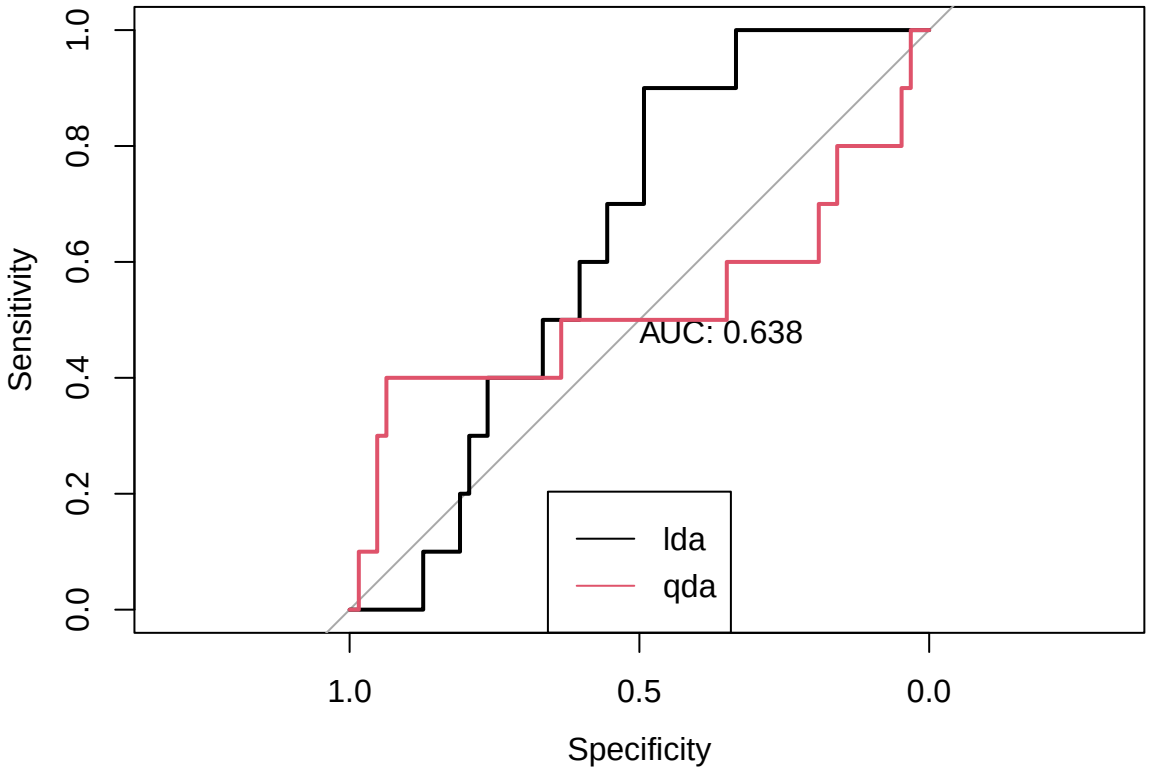
```
ROC_qda <- roc(data.test$failures, pred_qda)
```

```
## Warning in roc.default(data.test$failures, pred_qda): 'response' has more than
## two levels. Consider setting 'levels' explicitly or using 'multiclass.roc'
## instead
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls > cases
```

```
plot(ROC_lda, print.auc=TRUE,  print.auc.y = 0.5,xlim = c(1,0))
plot(ROC_qda,add = TRUE,col = 2)
legend('bottom', col=1:2, paste(c('lda', 'qda')),  lwd=1)
```



On remarque que les courbes ROC pour LDA et QDA sont assez différentes. En effet, l'aire sous la courbe pour LDA vaut 0.63 tandis que celle pour QDA vaut 0.52. Ainsi, les valeurs ne sont pas aussi élevés que celles qu'on avait en TP. De plus, on voit que le critère AUC pour LDA est meilleure que celui pour QDA ce qui peut être expliqué par le fait que les classes se chevauchent entre elles.

```
ROC_lda$auc
```

```
## Area under the curve: 0.6381
```

```
ROC_qda$auc
```

```
## Area under the curve: 0.5238
```

CART

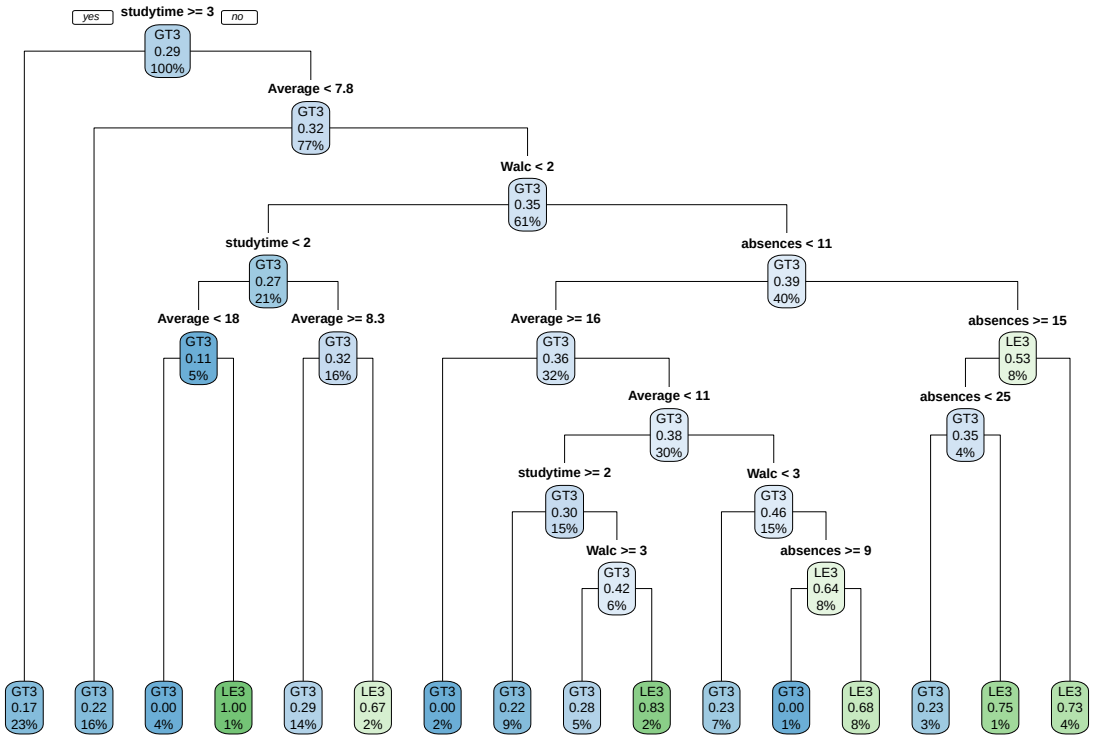
On étudie la probabilité pour un lycéen d'appartenir a une famille nombreuse ou non en fonction de différents critères. La variable à prédire est la variable qualitative famsize à deux modalités : LE3 :3 membres ou moins dans sa famille, GT3 : plus de 3 membres dans sa famille. Pour cela, on construit l'arbre CART

```
dataCart = data
dataCart = data[,-c(2,3,4,5,6,8,9,10,11,12,13,14,15,16,17,18,20)]
str(dataCart)
```

```
## 'data.frame':    395 obs. of  5 variables:
##  $ famsize  : Factor w/ 2 levels "GT3","LE3": 1 1 2 1 1 2 2 1 2 1 ...
```

```
##  $ studytime: int  2 2 2 3 2 2 2 2 2 2 ...
##  $ Walc     : int  1 1 3 1 2 2 1 1 1 1 ...
##  $ absences : int  6 4 10 2 4 10 0 6 0 0 ...
##  $ Average  : num  5.5 5 7.5 14.5 8 15 12 5.5 17 14.5 ...
```

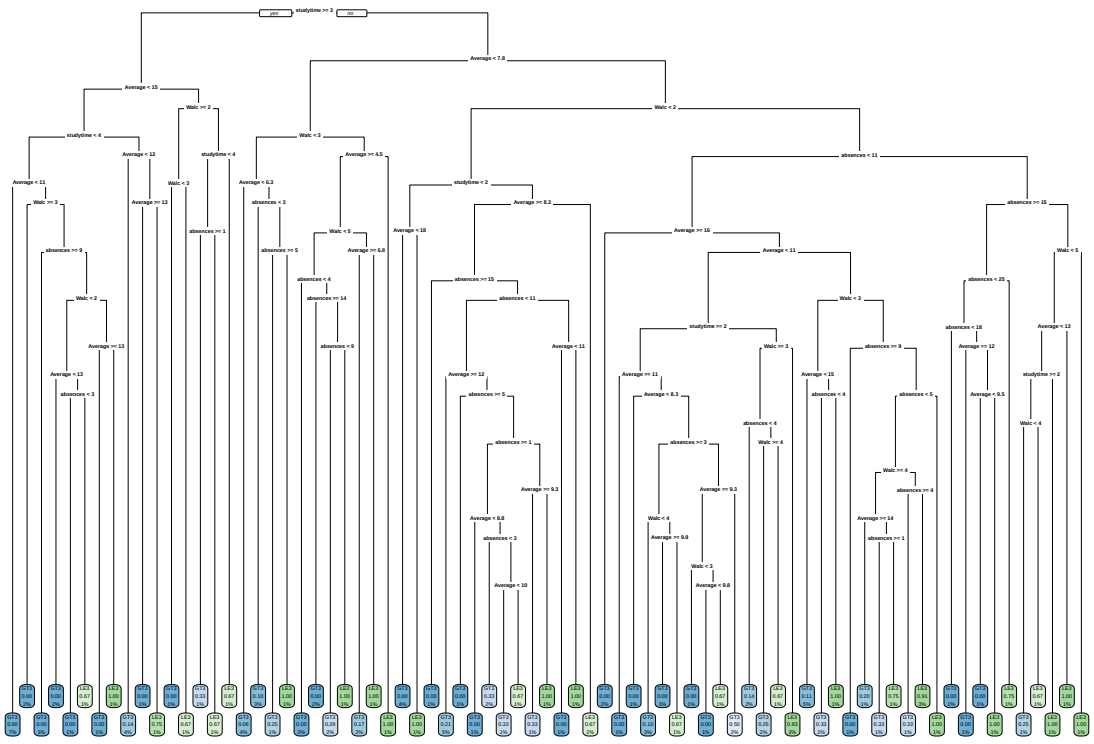
```
library(rpart)
library(rpart.plot)
arbre = rpart(famsize ~., dataCart,control = rpart.control(minsplit = 5))
#print(arbre)
rpart.plot(arbre,type = 1) #type = 1 pour mieux voir
```



On obtient un arbre plutôt cohérent. Un étudiant ayant un *studytime* > 3 à 17% d'appartenir à une famille nombreuse tandis qu'un étudiant qui s'absente beaucoup et boit souvent aura 73% de chance d'être dans une famille peu nombreuse. Évidemment, cela n'est pas du tout représentatif de la vie réelle.

Élagage On part de l'arbre le plus profond avec cp $= 0$

```
arbre.opt = rpart(famsize ~., dataCart,control = rpart.control(minsplit = 5,cp = 0))
rpart.plot(arbre.opt,type = 0)
```

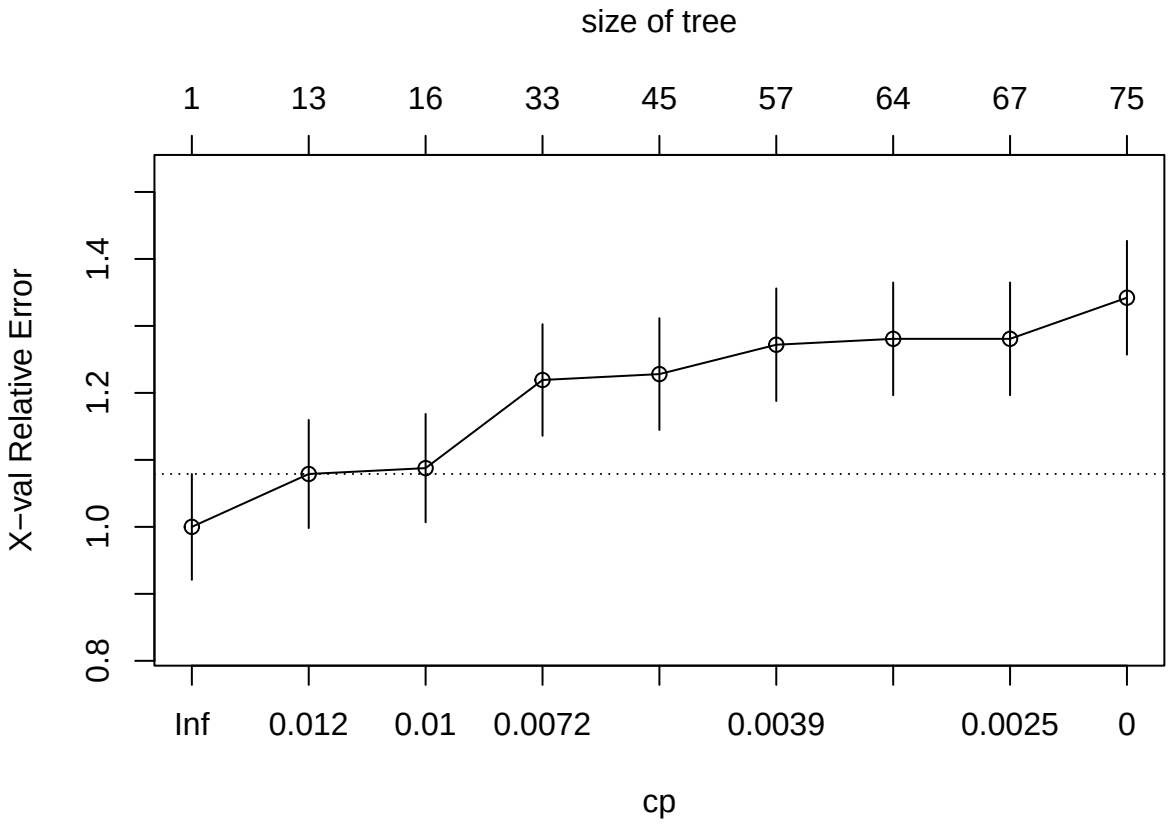


On cherche la valeur de xerror la plus petite, ici c'est 1 donc cp optimal = 0

```
set.seed(1)
printcp(arbre.opt)
```

```
##
## Classification tree:
## rpart(formula = famsize ~ ., data = dataCart, control = rpart.control(minsplit = 5,
##     cp = 0))
##
## Variables actually used in tree construction:
## [1] absences  Average   studytime Walc
##
## Root node error: 114/395 = 0.28861
##
## n= 395
##
##          CP nsplit rel error xerror     xstd
## 1 0.0122807      0   1.00000 1.0000 0.078995
## 2 0.0116959     12   0.78947 1.0789 0.080730
## 3 0.0087719     15   0.75439 1.0877 0.080908
## 4 0.0058480     32   0.60526 1.2193 0.083258
## 5 0.0043860     44   0.53509 1.2281 0.083393
## 6 0.0035088     56   0.48246 1.2719 0.084033
## 7 0.0029240     63   0.45614 1.2807 0.084154
## 8 0.0021930     66   0.44737 1.2807 0.084154
## 9 0.0000000     74   0.42982 1.3421 0.084928
```

```
plotcp(arbre.opt)
```



Prédiction

```
PredCartGood = data.frame(studytime = 4,Walc = 1,absences = 0,Average = 16.5)
PredCartBad = data.frame(studytime = 1,Walc = 4,absences = 40,Average = 7.5)
predGood= predict(arbre.opt, newdata=PredCartGood, type="prob")
predBad= predict(arbre.opt, newdata=PredCartBad, type="prob")
predGood
```

```
##         GT3       LE3
## 1 0.3333333 0.6666667
```

```
predBad
```

```
##   GT3 LE3
## 1   1   0
```

Bon élève prédit dans la famille peu nombreuse à 66%, et l'élève moins sérieux à lui 100% de chance d'avoir plus de 3 personnes dans sa famille. Ces conclusions ne sont pas représentatives bien-sûr, mais on peut toutefois se dire qu'un étudiant ayant peu de freres et soeur, a plus de temps pour travailler dans de bonnes conditions, etc... ce qui n'est pas forcèment le cas de quelqu'un vivant avec plus de 3 frères et soeurs.

Random Forest

```
dataRF = dataCart
str(dataRF)
```

```
## 'data.frame':    395 obs. of  5 variables:
##  $ famsize  : Factor w/ 2 levels "GT3","LE3": 1 1 2 1 1 2 2 1 2 1 ...
##  $ studytime: int  2 2 2 3 2 2 2 2 2 2 ...
##  $ Walc     : int  1 1 3 1 2 2 1 1 1 1 ...
##  $ absences : int  6 4 10 2 4 10 0 6 0 0 ...
##  $ Average  : num  5.5 5 7.5 14.5 8 15 12 5.5 17 14.5 ...
```

```
library(randomForest)
```

```
## randomForest 4.7-1
```

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

```
forest <- randomForest(famsize~.,dataRF)
forest
```

```
##
## Call:
##  randomForest(formula = famsize ~ ., data = dataRF)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 35.95%
## Confusion matrix:
##     GT3 LE3 class.error
## GT3 235  46   0.1637011
## LE3  96  18   0.8421053
```

```
p=ncol(dataRF)
sqrt(p)
```

```
## [1] 2.236068
```

On a 2 variables sélectionnes à chaque étapes.

```
print(forest)
```

```
##
## Call:
##  randomForest(formula = famsize ~ ., data = dataRF)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 35.95%
## Confusion matrix:
##     GT3 LE3 class.error
## GT3 235  46   0.1637011
## LE3  96  18   0.8421053
```

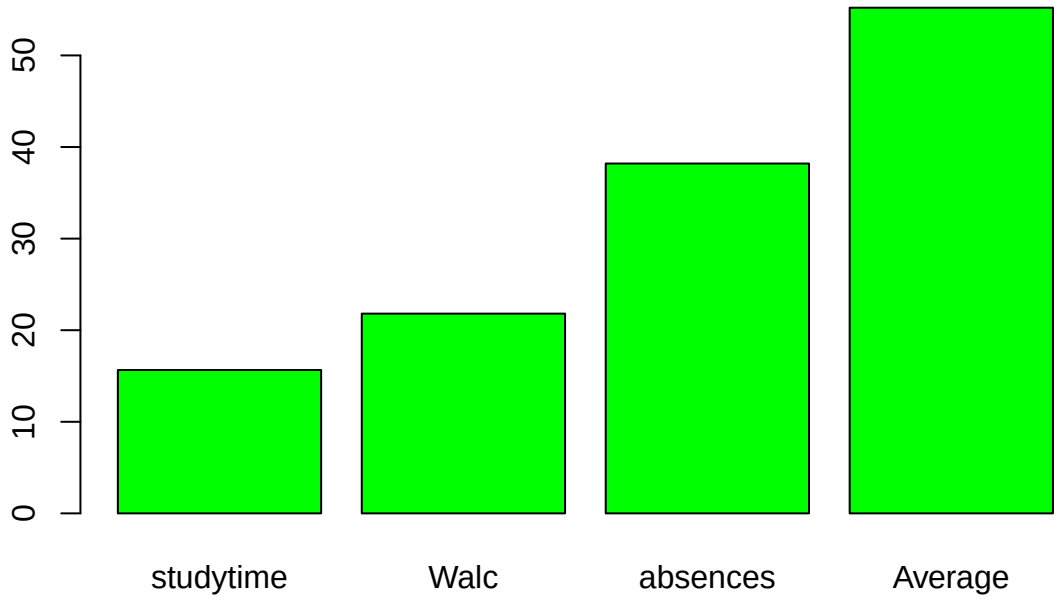
Importance des variables

```
importance = forest$importance
importance
```

```
##           MeanDecreaseGini
## studytime         15.65299
## Walc              21.80680
```

```
## absences          38.19807
## Average           55.21025
```

```
barplot(t(forest$importance),col="green")
```



Comparaison avec CART :

```
arbre.opt$variable.importance
```

```
##   Average  absences      Walc studytime
## 45.785918 35.125875 21.356475  9.104837
```

On trouve le même ordre d'importance pour CART et RandomForest.

Conclusion

Nous avons appliqué de nombreux modèles que ce soit la régression au 1er semestre puis la classification non supervisée avec **CAH** et **K-means** et enfin la classification supervisée avec les modèles d'analyses discriminantes (LDA,QDA), **ACP**, **CART** et **RandomForest**. Cela nous a permis de nous rendre compte que chaque jeu de données est différent et que tous les modèles ne fonctionnement pas toujours. Les jeu de données des séances de TP étaient conçus spécialement pour que les modèles fonctionnent "parfaitement".Dans notre cas à nous, notre jeu de données est exploitable pour du clustering par exemple mais pas pour de la régression comme on a pu le voir. Cela dit, se confronter à ce genre de problème permet par la suite de détecter plus rapidement les éventuelles incompatibilités du jeu de données ce qui est un atout important pour quelqu'un travaillant dans la data science et la statistique numérique.