

## S04: Statistiques multivariées

Analyse de données quantitatives avec R

Samuel Coavoux

- 1 Modèles de regression
- 2 Analyse géométrique de données
- 3 Clustering

# Modèles de regression

## Régression linéaire: lm

## Classe formule

La fonction `lm()` (linear models) permet d'ajuster des modèles de régression linéaire.

Elle prend comme premier argument un objet de classe **formule** ; la variable dépendante est précisée en premier, suivi d'un tilde (~), puis de l'interaction entre variables indépendantes.

- L'interaction est habituellement précisée par + (dans le modèle linéaire classique:  $Y = \alpha + \beta_1 X_1 + \beta_2 X_2$ ).
- On peut également ajouter, plutôt qu'une variable, l'interaction entre deux variables, en employant :.  $x \sim a:b$  revient à chercher  $Y = \alpha + \beta_1 X_1 X_2$  (surtout utile pour les régressions logistiques, lorsque l'on cherche l'interaction entre deux facteurs corrélés).
- Enfin, \* cherche à la fois l'addition et l'interaction.  $x \sim a*b$  est équivalent à  $x \sim a + b + a:b$ .

## Classe formule

On peut enfin transformer les variables directement dans une formule. Par exemple  $x \sim a + \log(b)$ . Si l'on souhaite utiliser un terme réservé pour la classe formule comme `+`, il faut l'enclore dans `I()`. Ainsi,  $x \sim a + b$  prend `a` et `b` comme variable indépendante, alors que  $x \sim I(a + b)$  prend **la somme de a et b** comme variable indépendante.

# lm()

Pour éviter d'avoir à répéter le nom du data.frame pour chaque variable de la formule, on peut employer l'argument data. Ainsi, les deux notations ci-dessous sont équivalentes

```
lm(imueclt ~ happy + income_dec, data=d)  
lm(d$imueclt ~ d$happy + d$income_dec)
```

L'argument weights permet de préciser un vecteur de pondération.

## Explorer un modèle

Par défaut, la méthode print des objets lm (`stats:::print.lm()`) donne assez peu d'informations : seulement les coefficients, et la commande employée pour produire le résultat. `summary.lm()` est beaucoup plus disert. On y obtient:

- un summary des résidus;
- les coefficients avec l'erreur standard, la valeur t et la p-value associée au test de nullité;
- quelques indicateurs de l'ajustement du modèle: R-squared, F, p-value;
- le nombre de valeurs manquantes

## Variabes

La variable dépendante doit être une variable numérique. lm() accepte des factor, mais c'est particulièrement déconseillée (en gros, la variable factor devrait être transformée en numérique, de sorte que votre variable dépendante sera discrète et prendra comme valeur 1 à k où k est le nombre de modalités).

Les variables indépendantes peuvent être des factors. Dans ce cas, la première modalité (le premier level) sera considéré comme la modalité de référence, et les coefficients des autres modalités sera calculé. Pour changer de modalité de référence rapidement (c'est à dire pour passer un level en premier level d'un factor sans avoir à réécrire `factor(x, levels=c(liste des levels))`), on peut employer `relevel()`

```
d$gndr <- relevel(d$gndr, ref = "Female")
```

## Explorer un modèle: print()

```
## R code block
## 
## Call:
## lm(formula = imueclt ~ happy + income_dec, data = d)
## 
## Coefficients:
## (Intercept)      happy    income_dec
##       3.2059     0.2471      0.1210
```

# Explorer un modèle: summary()

```
summary(l1)

##
## Call:
## lm(formula = imueclt ~ happy + income_dec, data = d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -6.8865 -1.5236  0.1907  1.7286  6.6731
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.205880  0.058305  54.98 <2e-16
## happy       0.247081  0.007542  32.76 <2e-16
## income_dec  0.120984  0.005067  23.88 <2e-16
```

## Valeurs manquantes

**Attention!** Par défaut, lm supprime les lignes de la base de données contenant une valeur manquante. On peut facilement se retrouver, dans une enquête par questionnaire, à faire des régressions sur quelques pourcents de l'échantillon si l'on ajoute trop de variables sans y prendre garde. Il convient donc de:

- limiter le nombre de variables;
- recoder en amont les NA autant que possible.

## Explorer un modèle: plot()

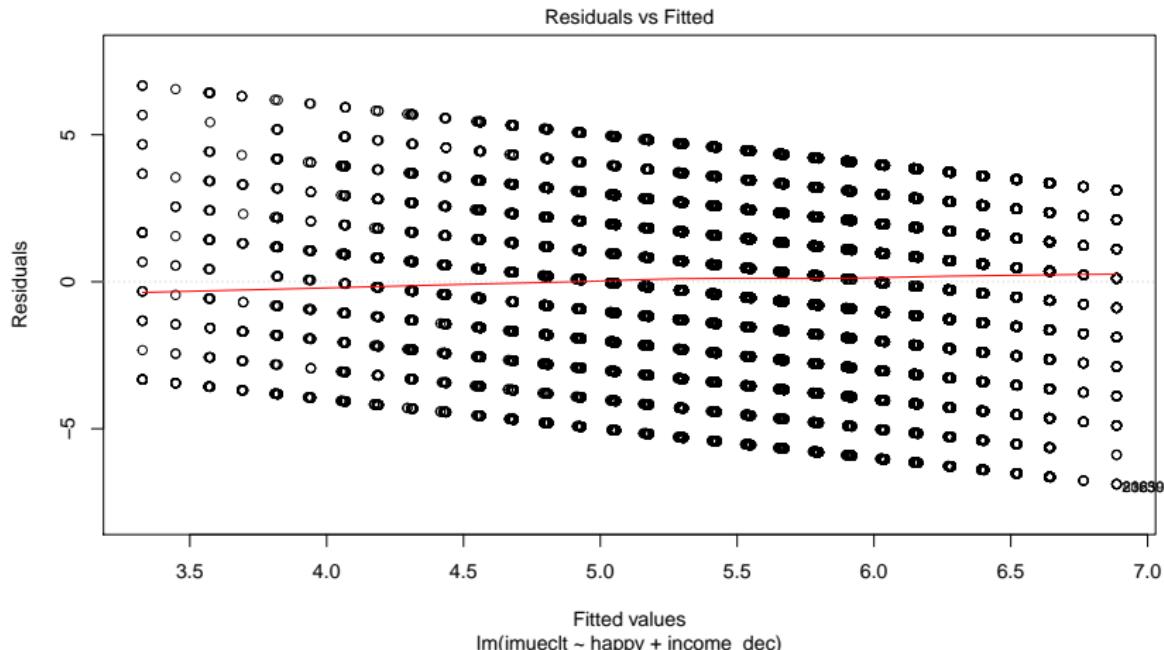
La fonction de base pour représenter graphiquement des modèles est la méthode `plot.lm()`. Par défaut, elle produit 4 graphiques (on peut en choisir un seul avec `which`):

- un scatterplot des résidu par valeur prédictive de  $Y$  (1);
- un diagramme Quantile-Quantile des résidu studentisé (2);
- un scatterplot de la racine des résidu studentisé par valeur prédictive de  $Y$  (3);
- un scatterplot des résidu studentisé pour les outliers (5).

Ces graphes devraient permettre de faire un premier diagnostic sur l'ajustement du modèle : vérifier la normalité des résidus et l'homoscédasticité du modèle.

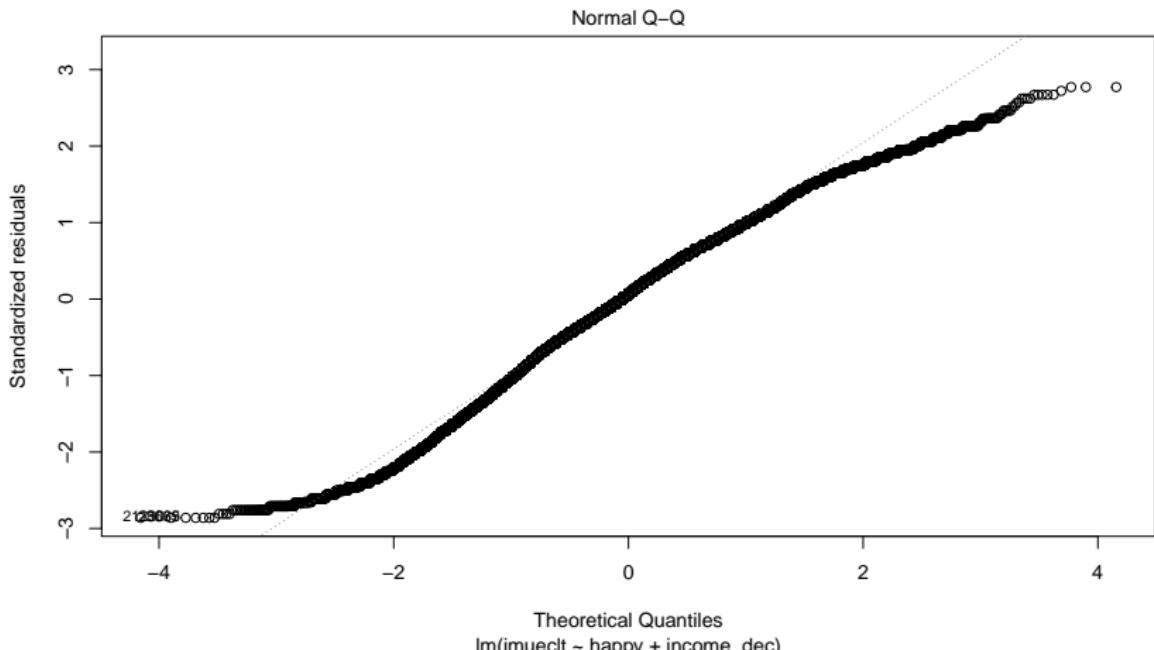
# Explorer un modèle: plot()

```
plot(l1, which = 1)
```



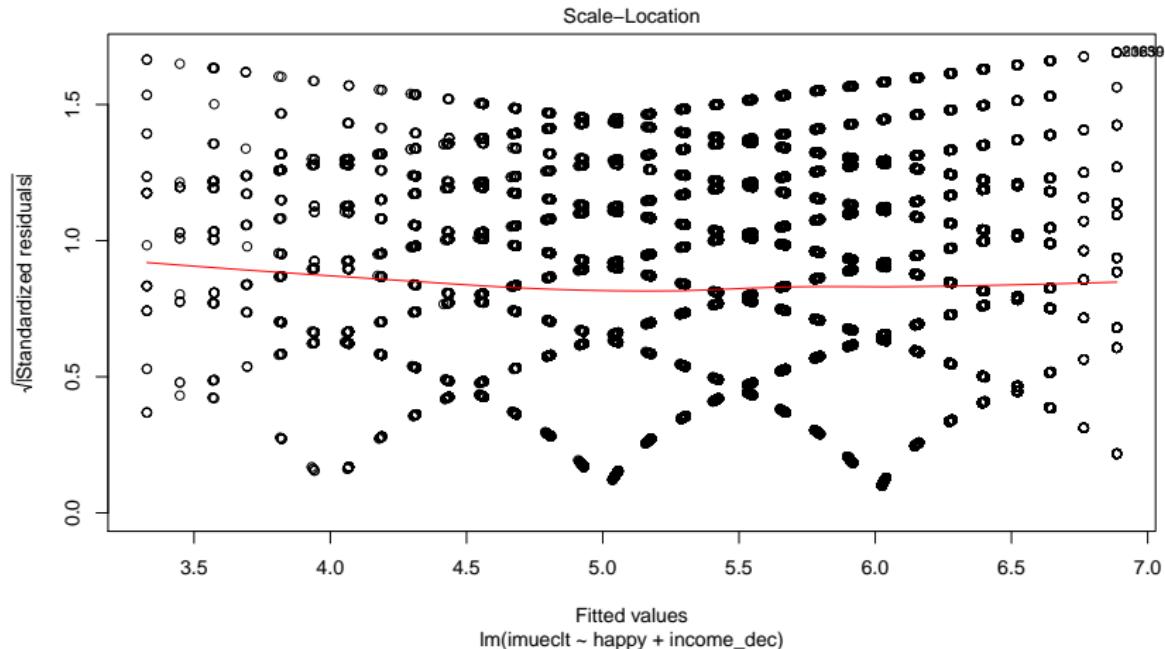
## Explorer un modèle: plot()

```
plot(l1, which = 2)
```



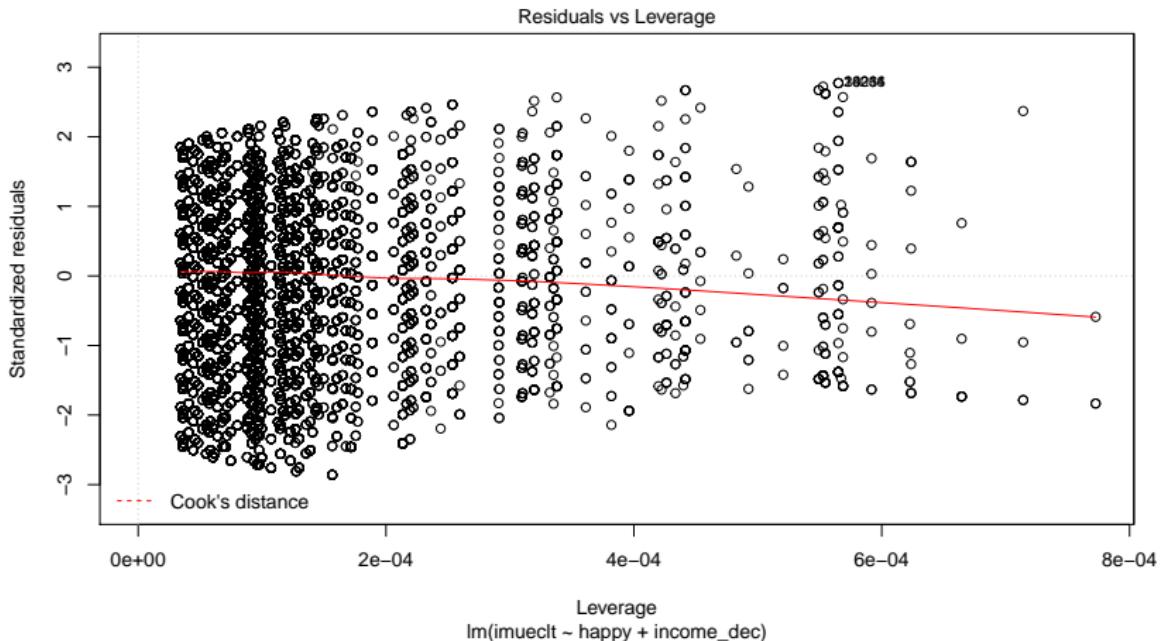
## Explorer un modèle: plot()

```
plot(l1, which = 3)
```



# Explorer un modèle: plot()

```
plot(l1, which = 5)
```



## Explorer un modèle: résultats de lm()

```
names(l1)
```

```
## [1] "coefficients"   "residuals"  
## [3] "effects"        "rank"  
## [5] "fitted.values"  "assign"  
## [7] "qr"              "df.residual"  
## [9] "na.action"       "xlevels"  
## [11] "call"            "terms"  
## [13] "model"
```

## Accéder aux coefficients

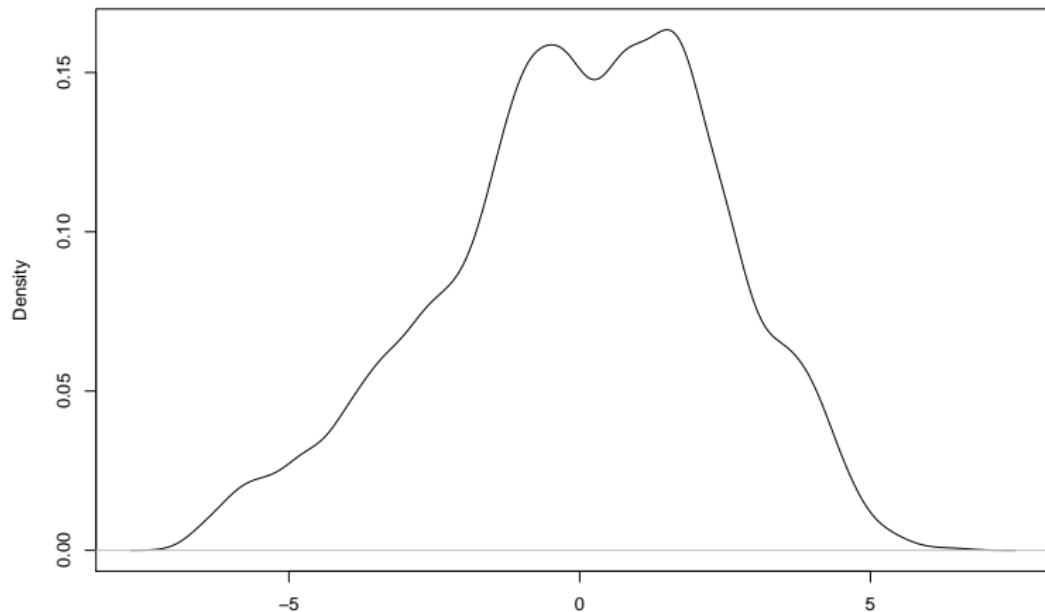
```
ll$coefficients
```

```
## (Intercept)      happy income_dec
## 3.2058797    0.2470806   0.1209840
```

## Accéder aux résidus

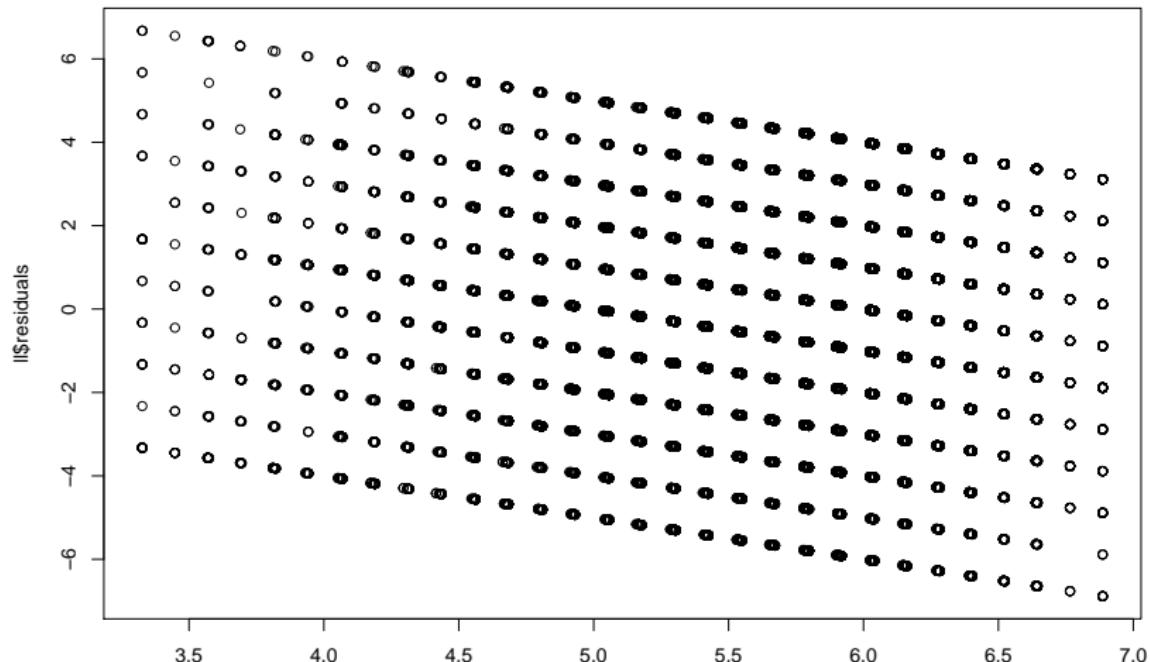
```
plot(density(l1$residuals))
```

```
density.default(x = l1$residuals)
```



## Accéder aux résidus

```
plot(l1$fitted.values, l1$residuals)
```



## Accéder aux valeurs prédites

```
residu <- ll$fitted.values -  
d$imueclt[complete.cases(d[, c("imueclt", "happy",  
"income_dec")])]
```

# ANOVA

# Anova

Pour réaliser une ANOVA, on doit d'abord ajuster un modèle linéaire.

```
ma <- anova(lm(imueclt ~ cntry, data = d))
```

# Anova

Contrairement à lm, summary() ne donne pas d'information intéressante, et il faut employer print() pour afficher les informations sur le test.

ma

```
## Analysis of Variance Table
##
## Response: imueclt
##           Df Sum Sq Mean Sq F value    Pr(>F)
## cntry      20 20372 1018.62 179.94 < 2.2e-16
## Residuals 38812 219714      5.66
##
## cntry     ***
## Residuals
## ---
```

## Régression logistique

## Modèle linéaire généralisé: glm()

La fonction `glm()` (Generalized Linear Model) permet de réaliser la plupart des régressions non-linéaires. Comme pour `lm()`, on donne le lien entre les variables dans une formule (dans laquelle on peut omettre le `data.frame` si l'on a précisé l'argument `data`), et on peut préciser un vecteur de pondération avec `weights`.

Par défaut, `glm()` produit une régression linéaire. Pour changer cela, on doit préciser l'argument `family` (préciser la fonction de répartition de l'erreur ainsi que le lien entre les termes du modèle). Par défaut, la famille `binomial` a pour argument de lien `link = "logit"`.

```
glm(Y ~ X1 + X2, data=mydata, family = "binomial")
# ce qui revient à:
glm(Y ~ X1 + X2, data=mydata, family = binomial(link =
                                         "logit"))
```

## Modèle linéaire généralisé: glm()

On ne voit dans ce cours que le modèle logit ; il suffit pour d'autres modèles de changer les arguments `family` et `link`. Voir l'aide de `family` pour cela.

```
?family
```

Par exemple, pour un modèle probit:

```
glm(Y ~ X1 + X2, data=mydata, family = binomial(link =  
"probit")
```

# Régression logistique

Pour une régression logistique, la variable dépendante doit être dichotomique. Elle peut être au format factor (alors, premier level = 0 et second level = 1 ; utiliser `relevel` pour le changer) ou numeric (0 ou 1). Si vous avez plus d'un niveau, **il n'y aura pas de message d'erreur** mais les coefficients n'auront pas grand sens.

Pour les modèles logit multinomiaux, cf. le package `mlogit`

# Régression logistique

```
titanic <- read.csv("data/titanic.csv",
                      stringsAsFactors = FALSE)
titanic$Children <- ifelse(titanic$Age < 18,
                           "Enfant",
                           "Adulte")
```

# Quels déterminants de la survie?

```
library(questionr)
lprop(table(titanic$Children, titanic$Survived))

##
##          0      1    Total
##  Adulte   61.4  38.6 100.0
##  Enfant   39.6  60.4 100.0
## Ensemble 58.6  41.4 100.0
```

# Quels déterminants de la survie?

```
lprop(table(titanic$Sex, titanic$Survived))
```

```
##  
##          0      1    Total  
##  female    33.3  66.7 100.0  
##  male     83.3  16.7 100.0  
## Ensemble 65.7  34.3 100.0
```

# Quels déterminants de la survie?

```
lprop(table(titanic$PClass, titanic$Survived))
```

```
##  
##          0      1    Total  
##          100.0  0.0 100.0  
##  1st      40.1  59.9 100.0  
##  2nd      57.3  42.7 100.0  
##  3rd      80.6  19.4 100.0  
## Ensemble 65.7  34.3 100.0
```

# Régression logistique

```
gt <- glm(Survived ~ Children + Sex + PClass,  
          data = titanic,  
          family = "binomial")
```

## Régression logistique: exploration

Comme pour lm, print donne peu d'information, et summary en donne beaucoup. plot.glm() produit les mêmes quatre graphiques que plot.lm()

```
summary(gt)
```

```
##  
## Call:  
## glm(formula = Survived ~ Children + Sex + PClass, family =  
##       data = titanic)  
##  
## Deviance Residuals:  
##      Min        1Q     Median        3Q       Max  
## -2.5991   -0.6638   -0.3799    0.7108    2.3085  
##  
## Coefficients:
```

## Régression logistique: odds ratio

Pour calculer les odds ratio, on prend l'exponentiel du coefficient:

```
exp(gt$coefficients)
```

```
##      (Intercept) ChildrenEnfant          Sexmale
##      8.04325743     3.51904273     0.07443514
##      PClass2nd       PClass3rd
##      0.41161922     0.12501288
```

# Ou alors:

```
# exp(coef(gt))
```

## Régression logistique: odds ratio

La fonction `odds.ratio` de `questionr` permet de le faire en ajoutant un intervalle de confiance (que l'on peut spécifier avec `level`)

```
odds.ratio(gt, level=.99)
```

```
## Waiting for profiling to be done...

##          OR 0.5 % 99.5 %      p
## (Intercept) 8.043 4.656 14.504 < 2.2e-16 ***
## ChildrenEnfant 3.519 1.707  7.430 9.746e-06 ***
## Sexmale       0.074 0.044   0.123 < 2.2e-16 ***
## PClass2nd     0.412 0.222   0.753 0.0001744 ***
## PClass3rd     0.125 0.066   0.230 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Analyse géométrique de données

# Packages

Il existe de nombreux packages pour réaliser des analyses géométriques de données. On utilisera FactoMineR, développé à Agrocampus Rennes.

Vous pouvez également employer:

- en base-R, `stats::factanal()` (analyse des correspondances) ou `stats::princomp()` (analyse en composantes principales)
- Autre packages: `ca`, `ade4` (package pour les sciences environnementales développé à Lyon 1)

# ACP

# Données

```
library(FactoMineR)
vars <- c(# actives
          "qfimedu", "qfimlng", "qfimchr",
          "qfimwht", "qfimwsk", "qfimcmt",
          "eduys", # Illus. quanti
          "gndr" # Illus. quali
          )
d_acp <- d[, vars]
```

# ACP

```
names(d_acp)
```

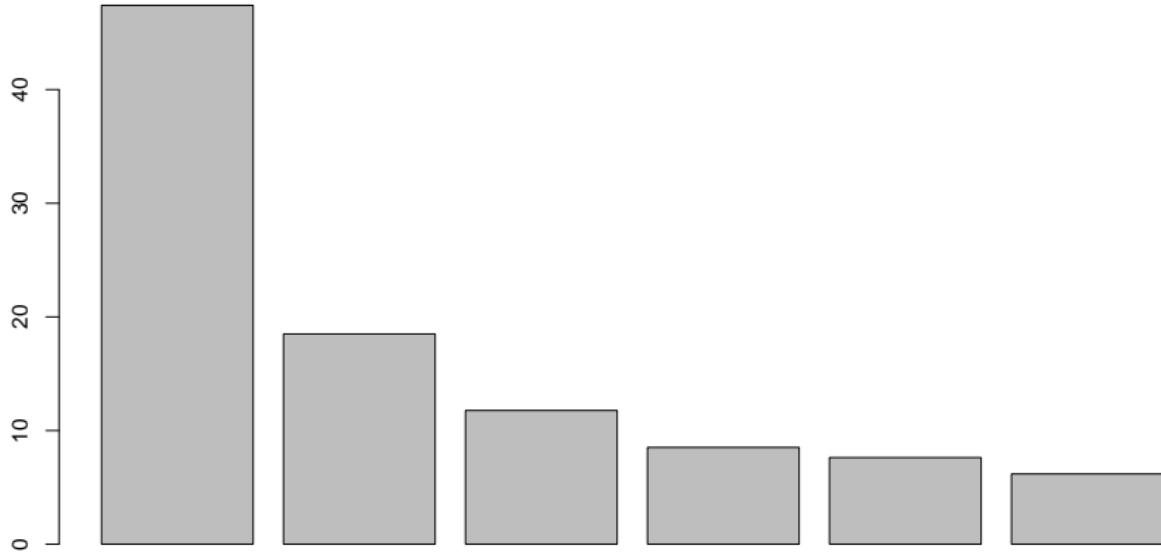
```
## [1] "qfimedu" "qfimlng" "qfimchr" "qfimwht"  
## [5] "qfimwsk" "qfimcmt" "eduys" "gndr"
```

```
qfi_acp <- PCA(d_acp, quanti.sup = 7, quali.sup = 8,  
                  graph = FALSE)
```

```
## Warning in PCA(d_acp, quanti.sup = 7, quali.sup  
## = 8, graph = FALSE): Missing values are imputed  
## by the mean of the variable: you should use the  
## imputePCA function of the missMDA package
```

# Eigenvalues

```
barplot(qfi_acp$eig$`percentage of variance`)
```



## Extraire les valeurs des variables

```
## Coordonnées et contribution de l'axe 1
cbind(qfi_acp$var$coord[, 1],
       qfi_acp$var$contrib[, 1])
```

```
##           [,1]      [,2]
## qfimedu 0.7308113 18.77215
## qfimlng 0.7190803 18.17433
## qfimchr 0.6139902 13.25032
## qfimwht 0.5924669 12.33763
## qfimwsk 0.8086383 22.98329
## qfimcmt 0.6418993 14.48229
```

## Extraire les valeurs des variables

```
## Coordonnées et contribution de l'axe 2
cbind(qfi_acp$var$coord[, 2],
      qfi_acp$var$contrib[, 2])
```

```
##           [,1]      [,2]
## qfimedu -0.3407394 10.460074
## qfimlng -0.3366885 10.212844
## qfimchr  0.6193270 34.556529
## qfimwht  0.6523270 38.337238
## qfimwsk -0.2017446  3.666856
## qfimcmt -0.1752335  2.766460
```

## Graphiques

L'intérêt de FactoMineR est d'avoir de bonnes méthodes de graph, avec beaucoup d'options. `plot()` renvoie à `plot.PCA()` pour un objet de classe PCA.

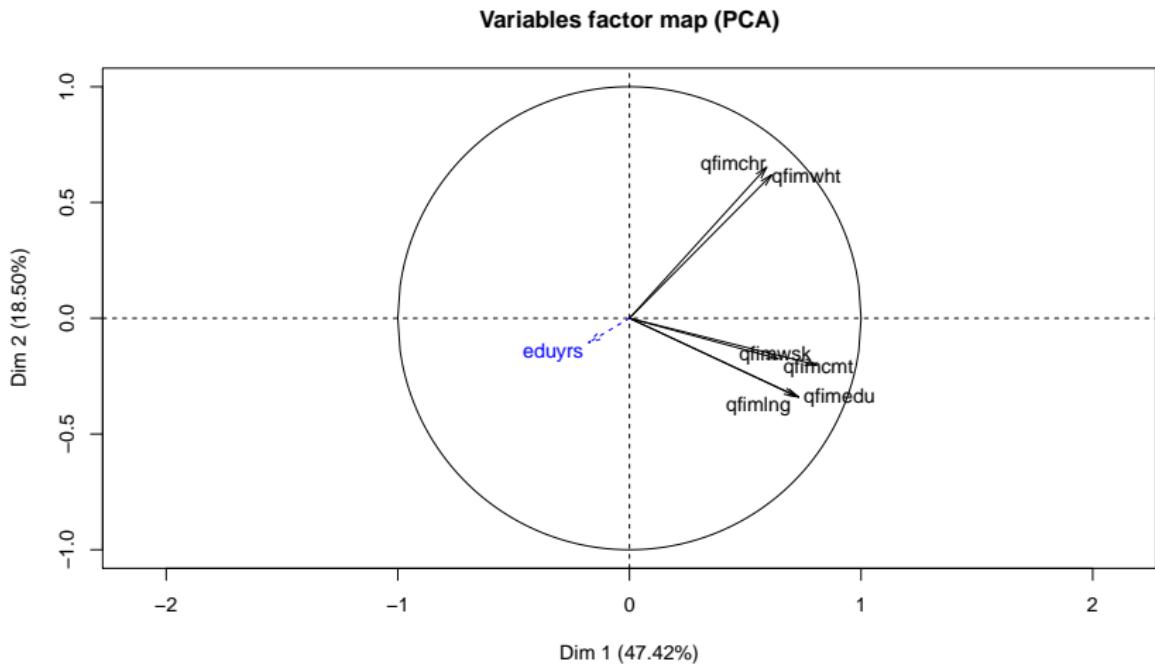
Par défaut, si on lance `PCA(d_acp, graph = TRUE)`, deux graphiques sont représentés : le nuage des individus, et le diagramme des variables, pour les deux premiers axes. C'est utile pour l'usage interactif ; dans les scripts, mieux vaut laisser `graph=FALSE` et produire ensuite explicitement les graphiques.

`plot.PCA` prend comme arguments:

- `x` = un objet de classe PCA
- `axes` = un vecteur numérique de taille 2 avec l'index des axes à représenter. Par défaut `c(1, 2)`: les deux premiers axes.
- `choix` = “ind” pour les individus, “var” pour les variables.

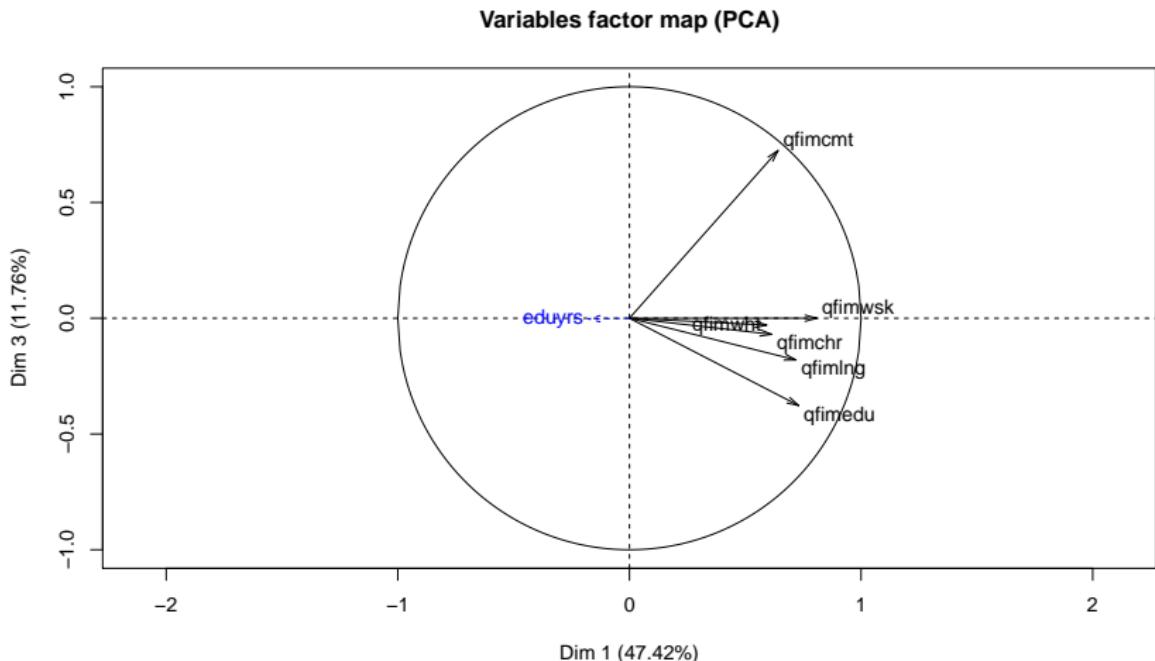
## Variables, axes 1 et 2

```
plot.PCA(qfi_acp, axes = c(1, 2), choix = "var")
```



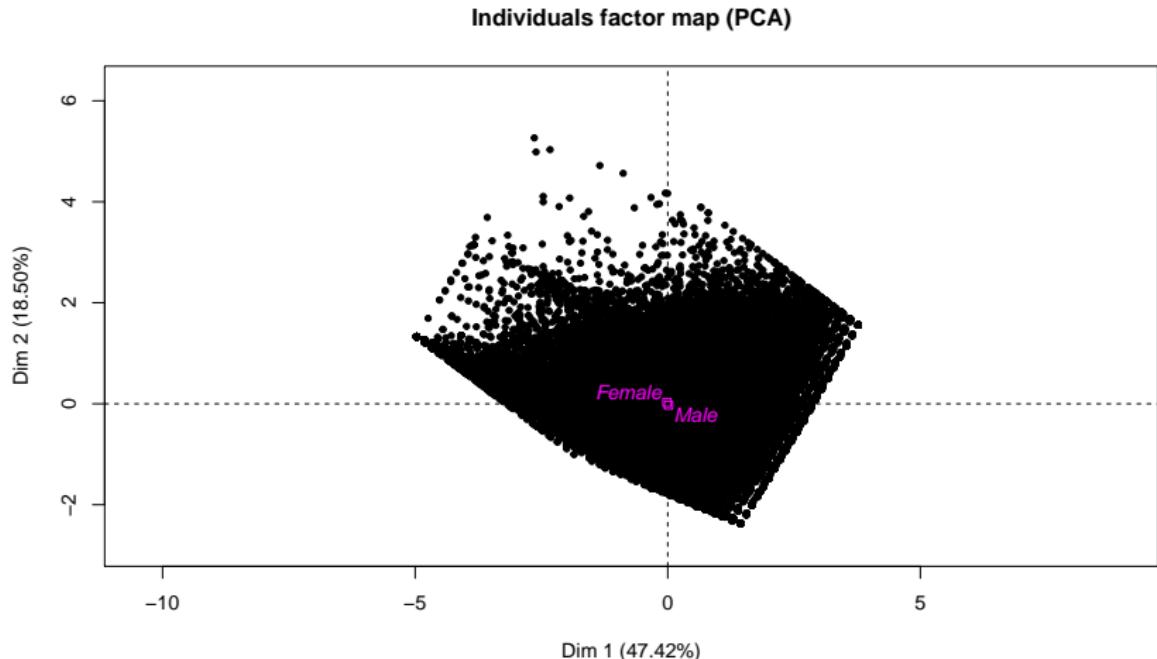
## Variables, axes 1 et 3

```
plot.PCA(qfi_acp, axes = c(1, 3), choix = "var")
```



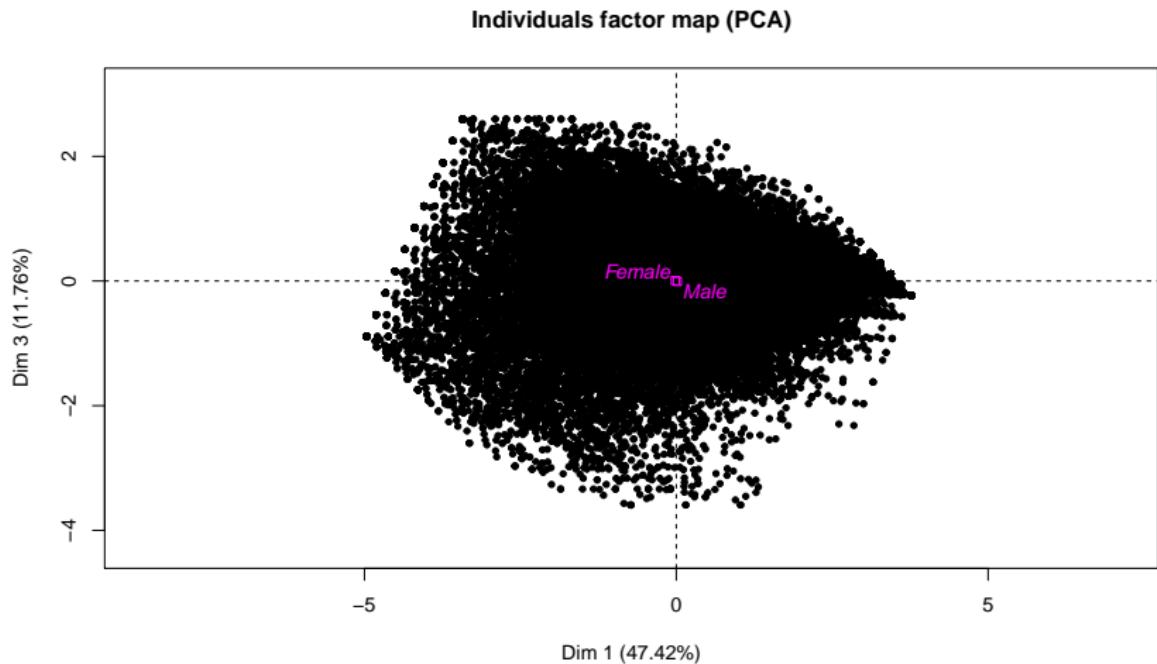
## Individus, axes 1 et 2

```
plot.PCA(qfi_acp, axes = c(1, 2), choix = "ind", label = "c")
```



## Individus, axes 1 et 3

```
plot.PCA(qfi_acp, axes = c(1, 3), choix = "ind", label = "c")
```



## Contrôle de ce qui apparaît

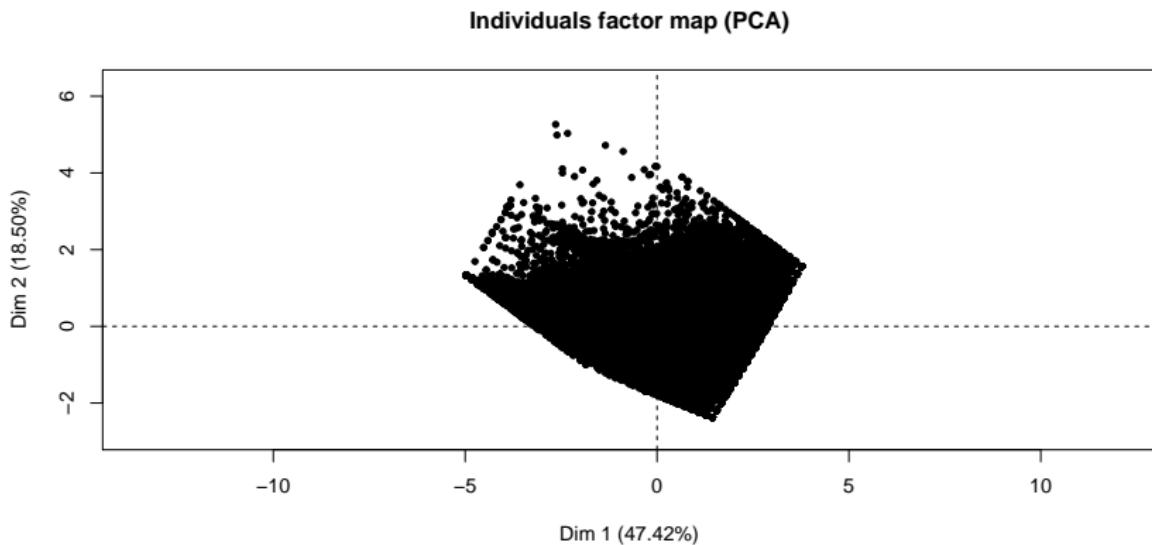
On peut limiter l'apparition des variables et individus de deux manières:

- avec `invisible`, on peut limiter l'apparition des individus (resp. variables) actifs ou illustratifs
- avec `select`, on peut ne sélectionner que certains individus/variables, sous condition, notamment, de leur contribution.

`Select` peut prendre pour valeur soit un vecteur numérique ou de nom (seuls les ind/var. correspondants seront affichés), ou une sélection par coordonnées, contribution, ou cosinus-carré. Dans ce cas, `select` est un vecteur character de taille 1 valant "paramètre nombre\_à\_afficher". Par exemple `param = "contrib 10"` affiche les 10 individus qui contribuent le plus au nuage sur les deux axes représentés ; `param = "cos2 20"` les 20 individus dont le

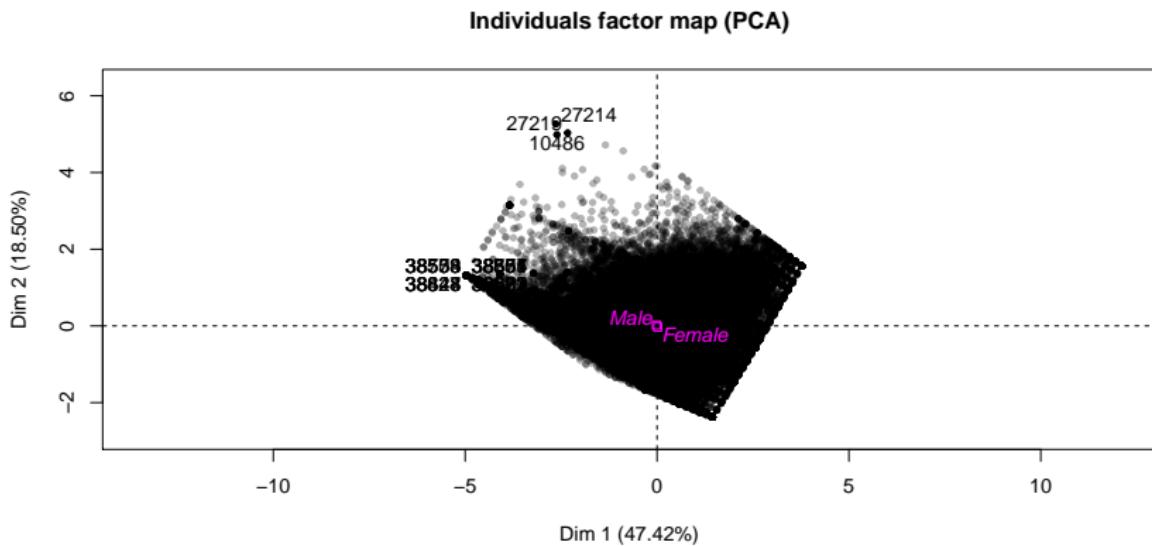
## Supprimer les variables qualitatives (illustratives)

```
plot.PCA(qfi_acp, axes = c(1, 2), choix = "ind",
          invisible = "quali", label = "none")
```



## Sélectionner par contribution

```
plot.PCA(qfi_acp, axes = c(1, 2), choix = "ind",
          select = "contrib 20")
```



## Valeurs manquantes

Par défaut, les fonctions de FactoMineR ignorent les valeurs manquantes (l'analyse est faite sur le sous-ensemble des individus pour lesquels toutes les variables sont renseignées).

`missMDA` est un package développé par les mêmes concepteurs que FactoMineR, qui inclut différentes méthodes pour imputer les valeurs manquantes dans un jeu de données, à partir d'analyses géométriques. Le package s'utilise en deux étapes: on commence par estimer le nombre de dimensions à prendre en compte avec `estim_ncpPCA()` (remplacer PCA par MCA en cas d'ACM) ; puis on impute les valeurs manquantes avec `imputePCA()` (resp. `imputeMCA`). Attention, il ne faut le faire que pour les variables actives.

# Valeurs manquantes

```
library(missMDA)
n <- estim_ncpPCA(d_acp[, 1:6])
complete_obs <- imputePCA(d_acp[, 1:6], ncp = n)
PCA(complete_obs)
```

## Généralisation

Les autres techniques, AFC (CA()) et ACM (MCA()), fonctionnent exactement de la même manière que PCA :

- réduire la base aux seules variables utiles ;
- imputer éventuellement les valeurs manquantes ;
- produire l'analyse avec la fonction correspondante et graph = FALSE ;
- graph des eigenvalues (\$eigen) ;
- analyse des coordonnées et contributions (\$var, \$ind pour PCA et MCA ; \$row et \$col pour CA) ;
- graphiques des axes retenus, pour les individus et pour les variables.

# CA

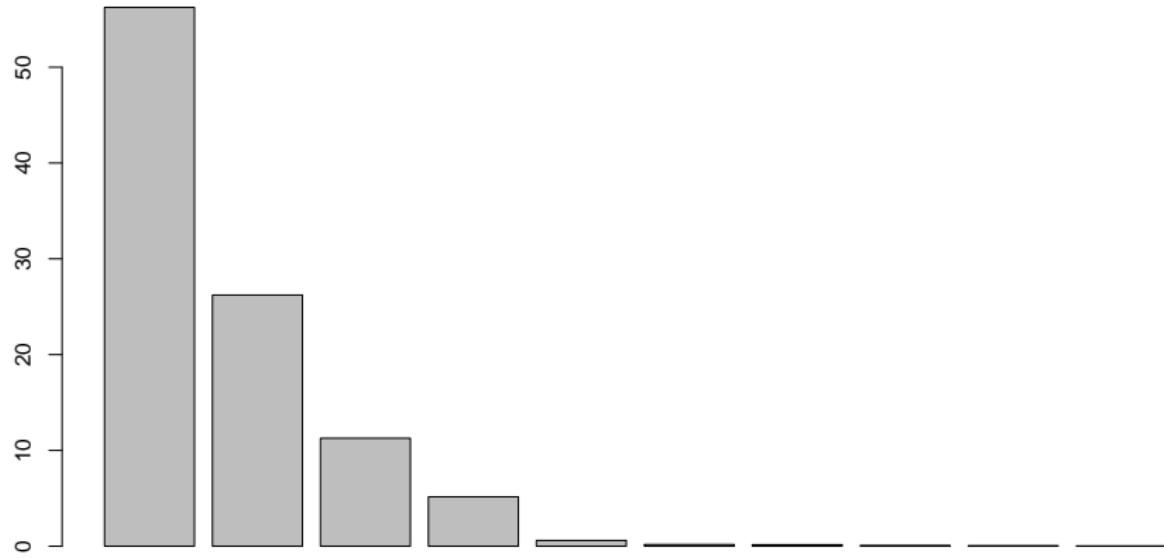
# Correspondance analysis

CA prend en premier argument un tableau de contingence.

```
rel_ca <- CA(table(d$cntry, d$rlgdnm),  
              graph = FALSE)
```

# Eigenvalues

```
barplot(rel_ca$eig$`percentage of variance`)
```



# Coordonnées et contribution des lignes

```
cbind(rel_ca$row$coord[, 1],  
      rel_ca$row$contrib[, 1])
```

```
##          [,1]      [,2]  
## AT -0.2047056  0.2253185  
## BE -0.1822871  0.1760811  
## CH -0.1943027  0.1732564  
## CZ -0.2712656  0.4734747  
## DE -0.2376710  0.5152438  
## DK -0.2351233  0.2487335  
## EE -0.2559541  0.4024974  
## ES -0.2433221  0.3414027  
## FI -0.2594101  0.4206969  
## FR -0.1541224  0.1364039  
## GB -0.2181846  0.3228478
```

## Coordonnées et contribution des lignes

```
cbind(rel_ca$row$coord[, 2],  
      rel_ca$row$contrib[, 2])
```

```
##           [,1]      [,2]  
## AT -0.63388722 4.635435496  
## BE -0.10351903 0.121834507  
## CH  0.13438043 0.177800460  
## CZ  0.22001133 0.668230747  
## DE  0.32408242 2.055418875  
## DK  0.96222274 8.937648124  
## EE  0.53145622 3.723083256  
## ES -0.64503733 5.147573180  
## FI  0.96070615 12.379573355  
## FR -0.26309590 0.852810181  
## GB  0.59324934 5.120979505
```

## Coordonnées et contribution des colonnes

```
cbind(rel_ca$col$coord[, 1],  
      rel_ca$col$contrib[, 1])
```

```
##                                     [,1]  
## Roman Catholic                 -0.2569369  
## Protestant                      -0.2569734  
## Eastern Orthodox                 -0.1871549  
## Other Christian denomination   -0.1451424  
## Jewish                          3.7544111  
## Islamic                         1.3721242  
## Eastern religions                -0.2442751  
## Other non-Christian religions  0.2136957  
## Not applicable                  -0.2526786  
## Refusal                         -0.2011613  
## No answer                       -0.2601058
```

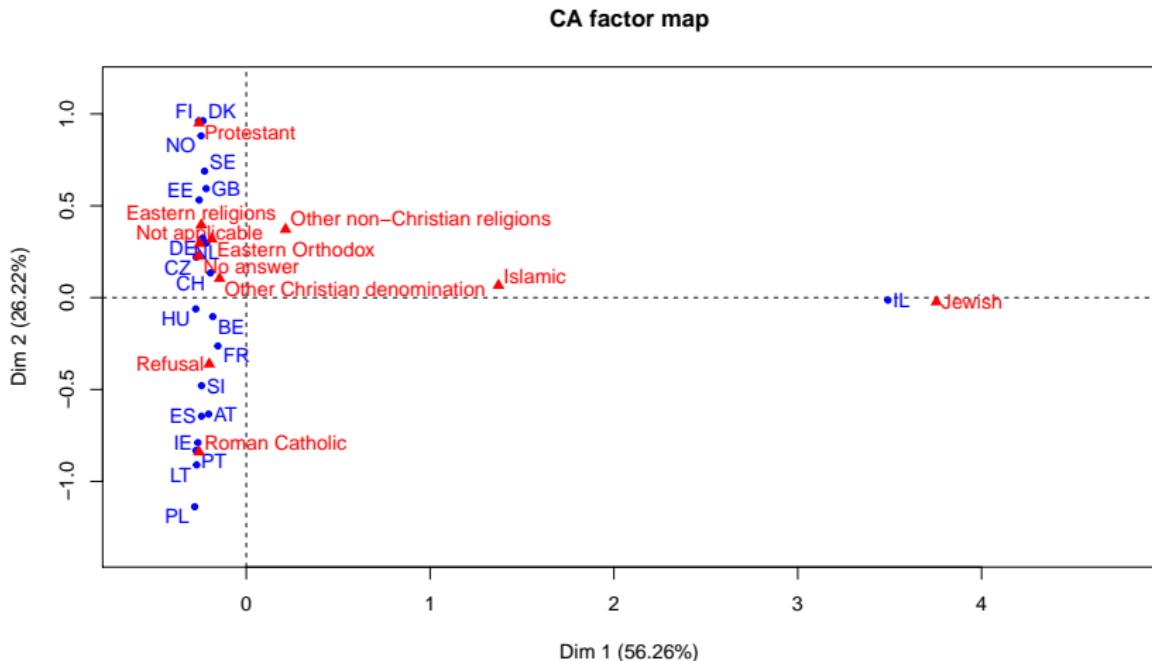
## Coordonnées et contribution des colonnes

```
cbind(rel_ca$col$coord[, 2],  
      rel_ca$col$contrib[, 2])
```

	[,1]
##	
## Roman Catholic	-0.83982954
## Protestant	0.95059553
## Eastern Orthodox	0.31882755
## Other Christian denomination	0.10411549
## Jewish	-0.02314617
## Islamic	0.06670883
## Eastern religions	0.39585253
## Other non-Christian religions	0.37128146
## Not applicable	0.29441890
## Refusal	-0.36138665
## No answer	0.22583394

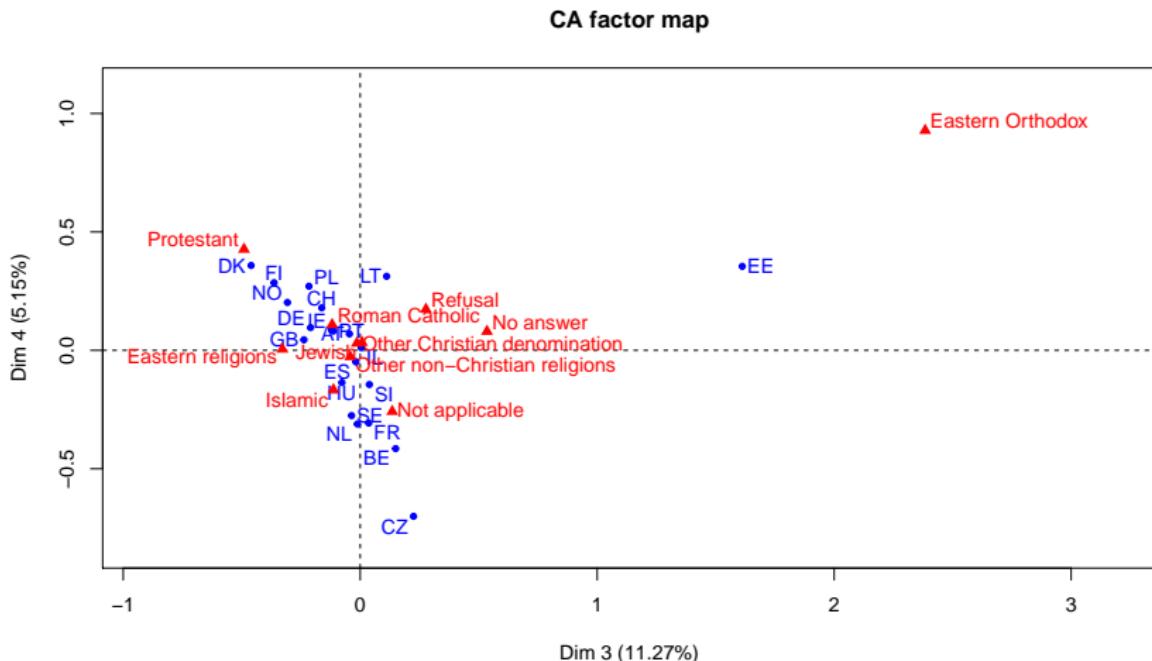
# Diagramme, axes 1 et 2

```
plot.CA(rel_ca, axes = c(1, 2))
```



## Diagramme, axes 3 et 4

```
plot.CA(rel_ca, axes = c(3, 4))
```



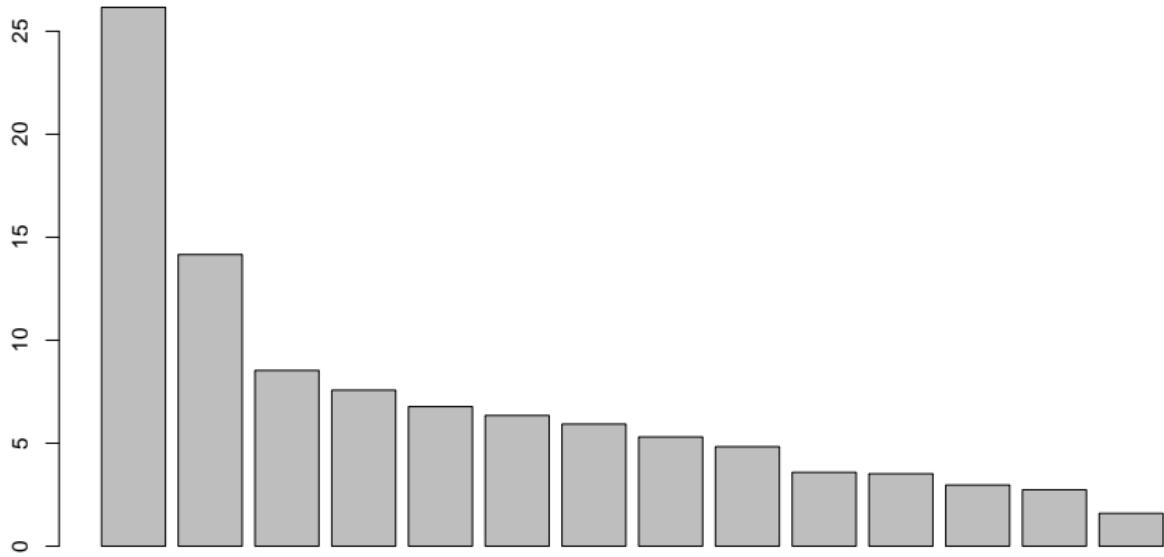
ACM

# ACM

```
d_acm <- d[, c("trstprl", "trstlgl", "trstpplc",
                 "trstpplt", "trstpprt", "trststep",
                 "trstun", "gndr", "cntry")]
d_acm <- d_acm[complete.cases(d_acm), ]
trust_acm <- MCA(d_acm, quali.sup = 8:9, graph = FALSE)
```

# Eigenvalues

```
barplot(trust_acm$eig$`percentage of variance`)
```



## Contribution et coordonnées

```
cbind(trust_acm$var$coord[, 1],  
      trust_acm$var$contrib[, 1])
```

```
##                                     [,1]      [,2]  
## trstprl_Weak      -0.91605805 8.43019793  
## trstprl_Average   0.14770888 0.20684777  
## trstprl_High      1.00329668 7.82803665  
## trstlgl_Weak      -1.03199439 7.35000363  
## trstlgl_Average   -0.15244262 0.19740706  
## trstlgl_High      0.70689010 5.94996114  
## trstplc_Weak      -1.14577368 5.32231144  
## trstplc_Average   -0.42913693 1.34083525  
## trstplc_High      0.48656986 3.78048283  
## trstplt_Weak      -0.71988570 7.47317260  
## trstplt_Average   0.57276056 2.99799567
```

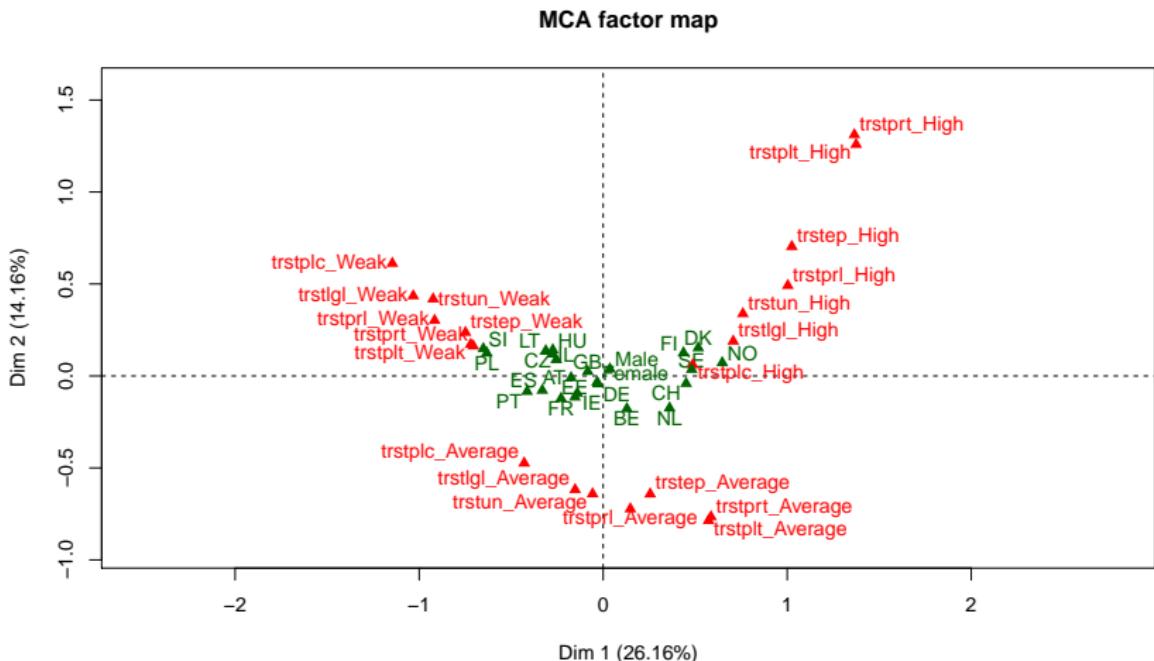
## Contribution et coordonnées

```
cbind(trust_acm$var$coord[, 2],  
      trust_acm$var$contrib[, 2])
```

```
##                                     [,1]      [,2]  
## trstprl_Weak      0.30189761  1.6920296  
## trstprl_Average -0.72283691  9.1540886  
## trstprl_High       0.49122554  3.4677901  
## trstlgl_Weak      0.43512146  2.4146306  
## trstlgl_Average -0.61828785  6.0010617  
## trstlgl_High       0.18890006  0.7851838  
## trstplc_Weak      0.61180131  2.8042726  
## trstplc_Average -0.47388277  3.0214979  
## trstplc_High        0.06074132  0.1088735  
## trstplt_Weak       0.17170218  0.7856450  
## trstplt_Average -0.78643549 10.4449848
```

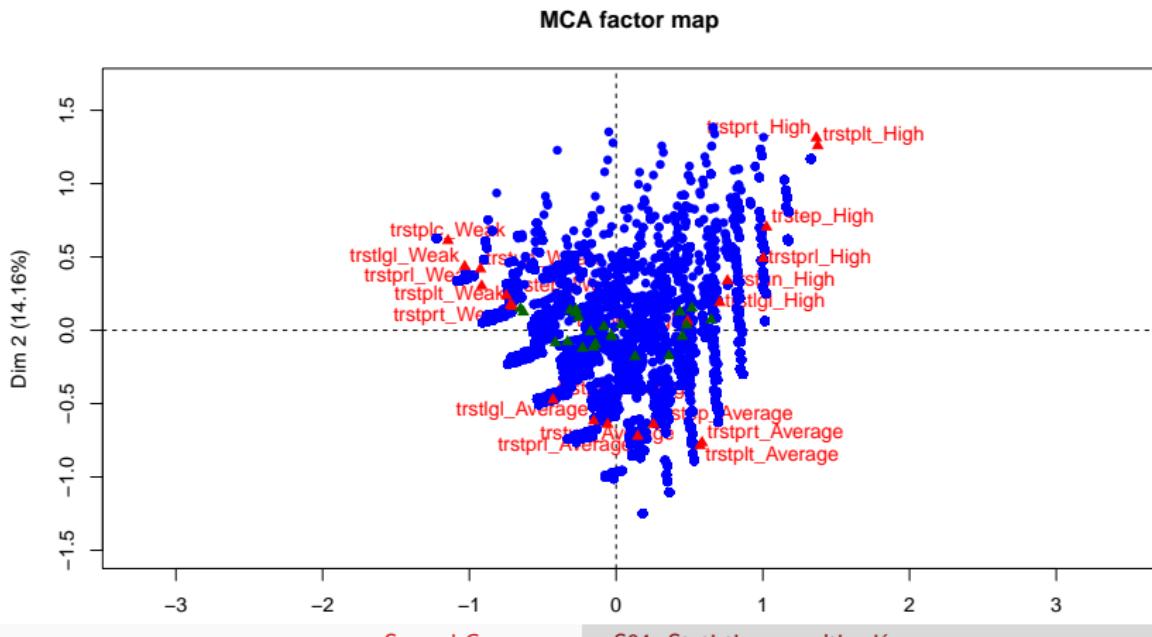
# Diagramme des modalités

```
plot.MCA(trust_acm, choix = "ind", invisible = "ind")
```



# Nuage des individus

```
plot.MCA(trust_acm, choix = "ind",  
         label = "var")
```



# Clustering

## À partir d'une AGD: HCPC

# HCPC

FactoMineR inclut une fonction de clustering spécialement conçue pour catégoriser à partir des résultats d'une analyse factorielle. HCPC (Hierarchical Clustering on Principle Components) classifie les individus (pour une analyse factorielle des correspondance, les lignes du tableau de contingence) à partir d'un algorithme agnes (agglomerative nesting).

HCPC fonctionne avec les trois types d'objets.

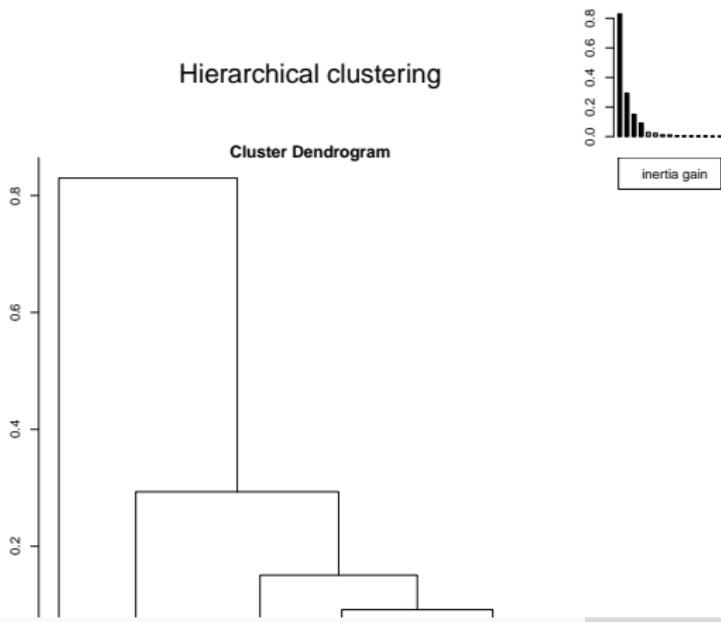
## HCPC: interactif

Par défaut, si l'on ne spécifie pas un nombre de clusters, HCPC() commence par représenter le dendrogramme et demander à l'utilisateur de cliquer à l'endroit où il souhaite couper, en indiquant par une ligne horizontale le lieu optimal.

Il vaut mieux éviter cela. En effet, cela ne fonctionnera que dans un usage interactif ; si l'on a un usage programmatique, mieux vaut spécifier manuellement le nombre de cluster. On empêche ce comportement en spécifiant `graph = FALSE` et/ou en spécifiant un nombre de cluster avec `nb.clust`. Comme pour les autres AGD, `graph = FALSE` est la meilleure option ; on choisit alors le nombre optimal de clusters proposé par l'algorithme, et l'on peut toujours représenter le dendrogramme par la suite.

# HCPC

```
rel_hcpc <- HCPC(rel_ca, graph = FALSE)  
plot.HCPC(rel_hcpc, choice = "tree")
```



# HCPC

À la vue du dendrogramme, on peut toujours souhaiter, a posteriori, couper l'arbre à un autre endroit.

```
rel_hcpc <- HCPC(rel_ca, graph = FALSE, nb.clust = 3)
```

## HCPC: récupérer les clusters

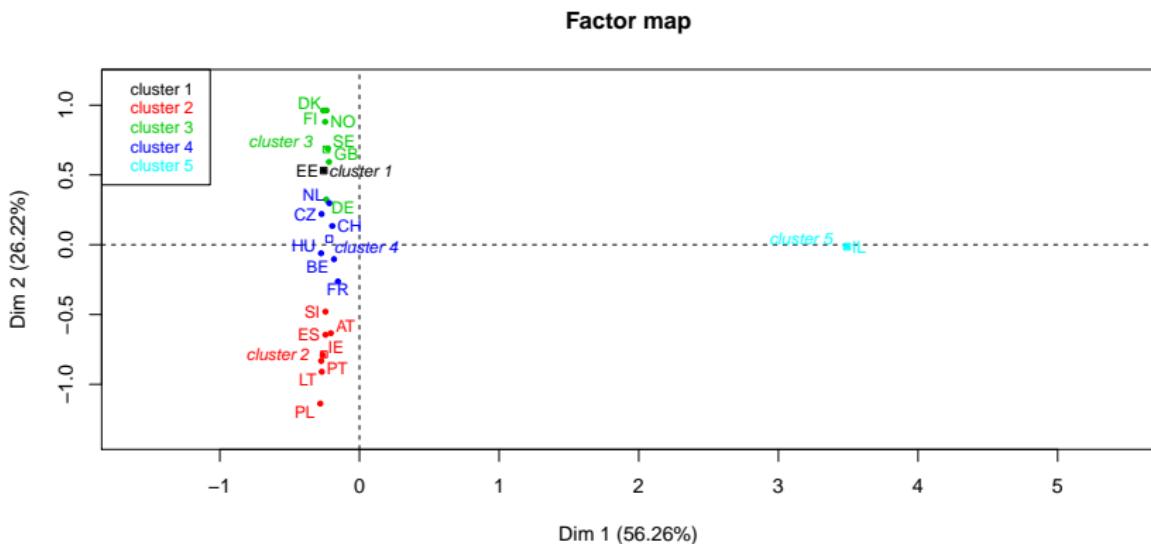
`rel_hcpc$data.clust` contient les données originales, auxquelles se sont ajoutés les clusters ; on peut alors les profiler en faisant, par exemple, des tableaux de contingence des variables supplémentaires par la variable `clust`.

```
rel_hcpc$data.clust$clust
```

```
## [1] 2 4 2 4 2 2 3 1 3 2 2 3 3 3 3 4 2 4 4 4 5
## Levels: 1 2 3 4 5
```

# HCPC: enrichir la représentation de l'AGD

```
plot.HCPC(rel_hcpc, axes = c(1, 2), choice = "map",  
          draw.tree = FALSE)
```



## HCPC: plus de détails

HCPC s'appuie sur la fonction `cluster::agnes()` ; cf. l'aide pour plus de détails. Vous pouvez notamment adapter la méthode et la métrique de calcul de la distance entre individus. On peut également utiliser `agnes()` pour faire des CAH directement sur un `data.frame`, sans passer par l'AGD.

NB: une CAH s'appuie sur une matrice de dissimilarité de  $n$  lignes et  $n$  colonnes (pour  $n =$  nombre d'individus classifiés). Par conséquent, le temps de calcul et la puissance de l'ordinateur, en particulier en termes de RAM, peut devenir rapidement très important. (ici, ne pas essayer de faire HCPC sur les résultats de l'ACM ou de l'ACP précédentes).