

Séance 2 : Statistiques multivariées

Introduction à la sociologie quantitative, niveau 2

Samuel Coavoux

1 Modèles de regression

2 Analyse géométrique de données

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 ~ 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 ~ a*b est équivalent à x ~ 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

Variables

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()

```
ll <- lm(imueclt ~ happy + income_dec, data=d)
print(ll)
```

```
##
## Call :
## lm(formula = imueclt ~ happy + income_dec, data = d)
##
## Coefficients :
## (Intercept)      happy    income_dec
##           3.2078      0.2470      0.1208
```

Explorer un modèle : summary()

```
summary(l1)
```

```
##  
## Call :  
## lm(formula = imueclt ~ happy + income_dec, data = d)  
##  
## Residuals :  
##      Min       1Q   Median       3Q      Max  
## -6.8859 -1.5235  0.1894  1.7289  6.6714  
##  
## Coefficients :  
##                 Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 3.207782  0.058323  55.00 <2e-16  
## happy       0.247009  0.007543  32.75 <2e-16  
## income_dec 0.120800  0.005068  23.83 <2e-16  
##  
## (Intercept) ***  
## happy       ***  
## ...
```

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()

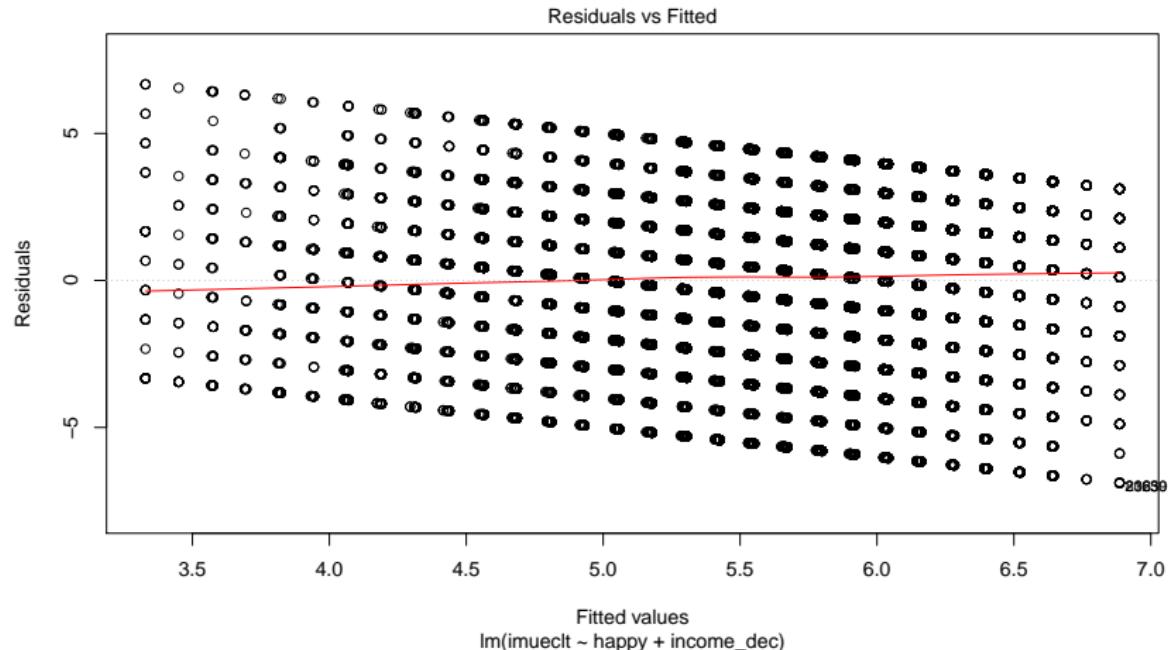
La fonction de base pour représenter graphiquement des modèles est la méthode `plot.lm()`. Par défaut, elle produit 4 graphiques (on peut en choisir un seul avec `which`):

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

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

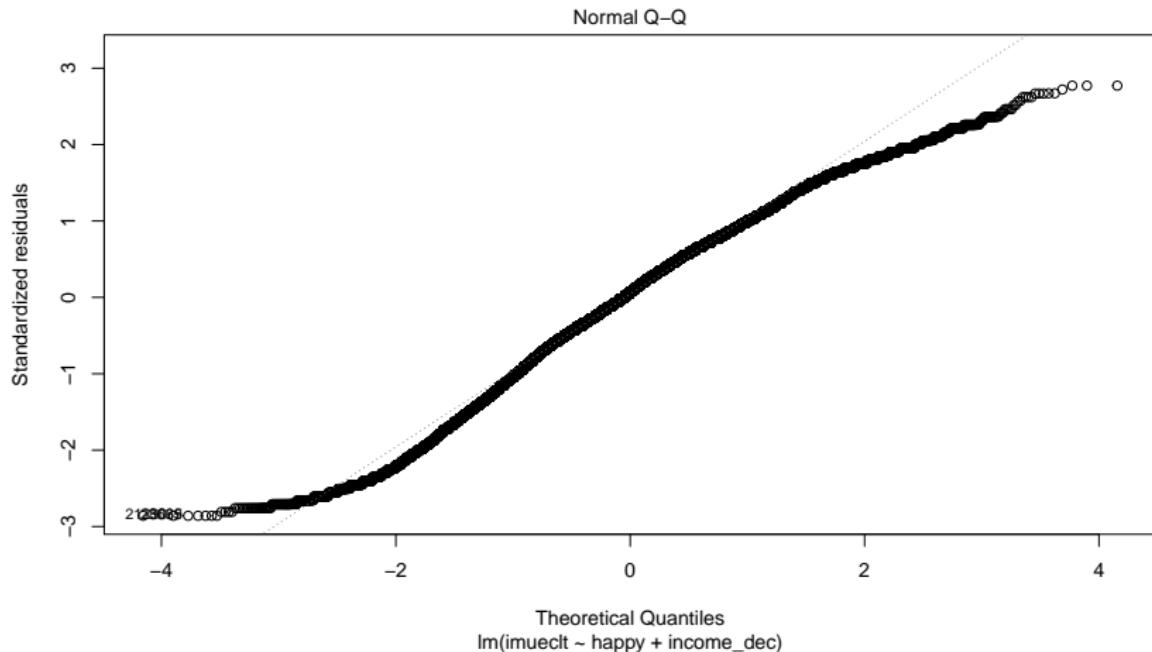
Explorer un modèle : plot()

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



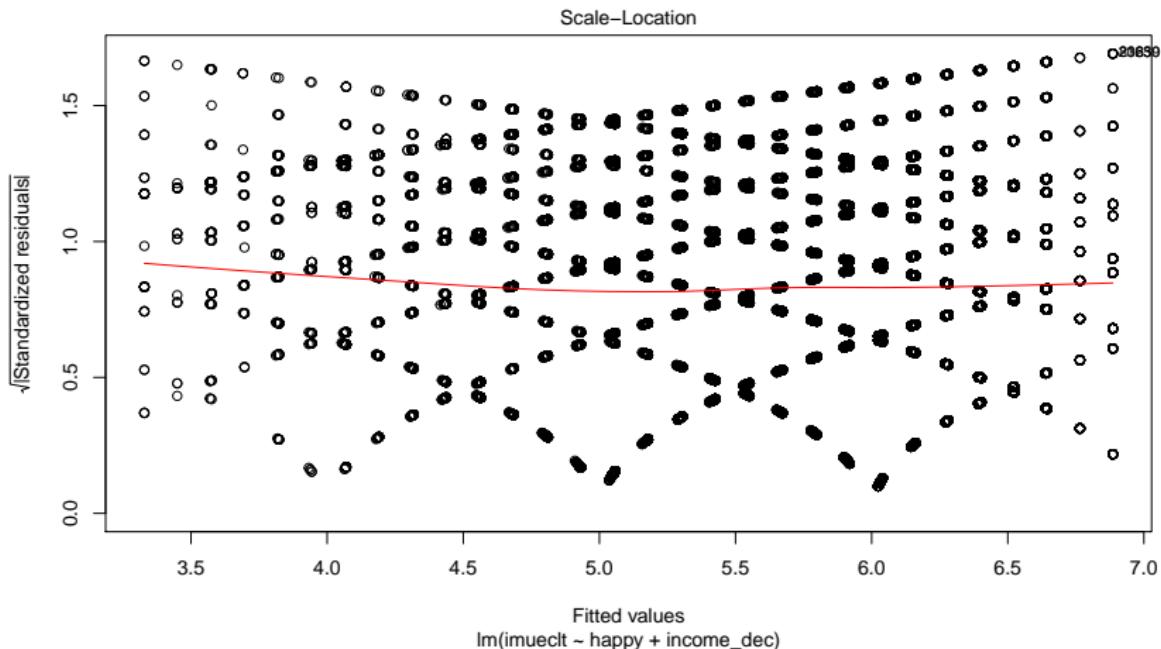
Explorer un modèle : plot()

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



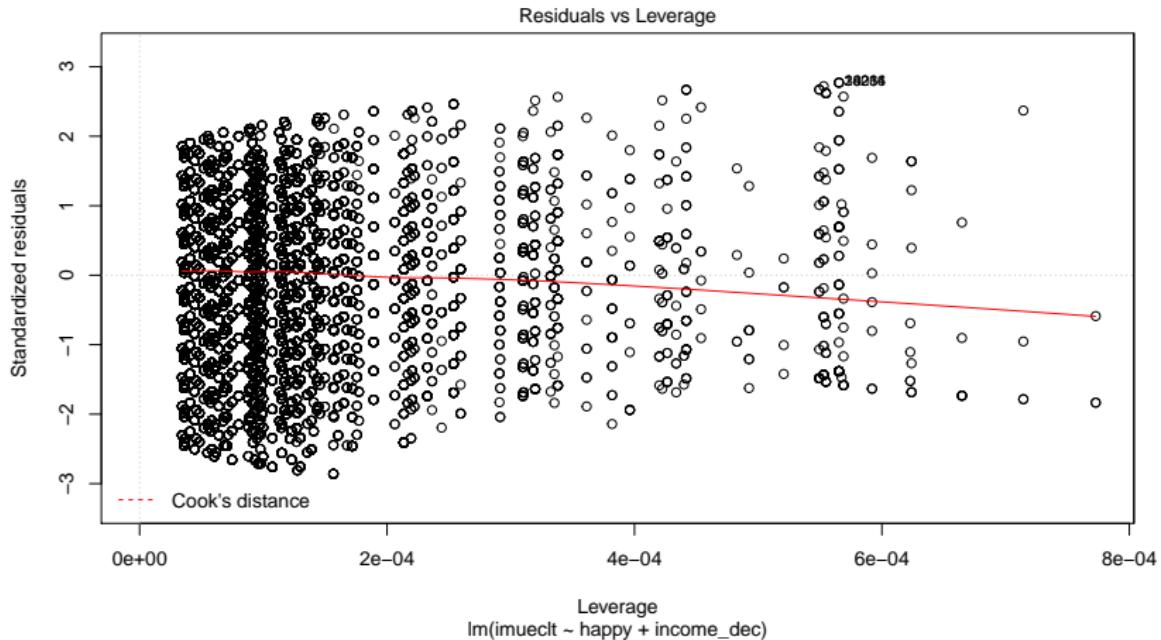
Explorer un modèle : plot()

```
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

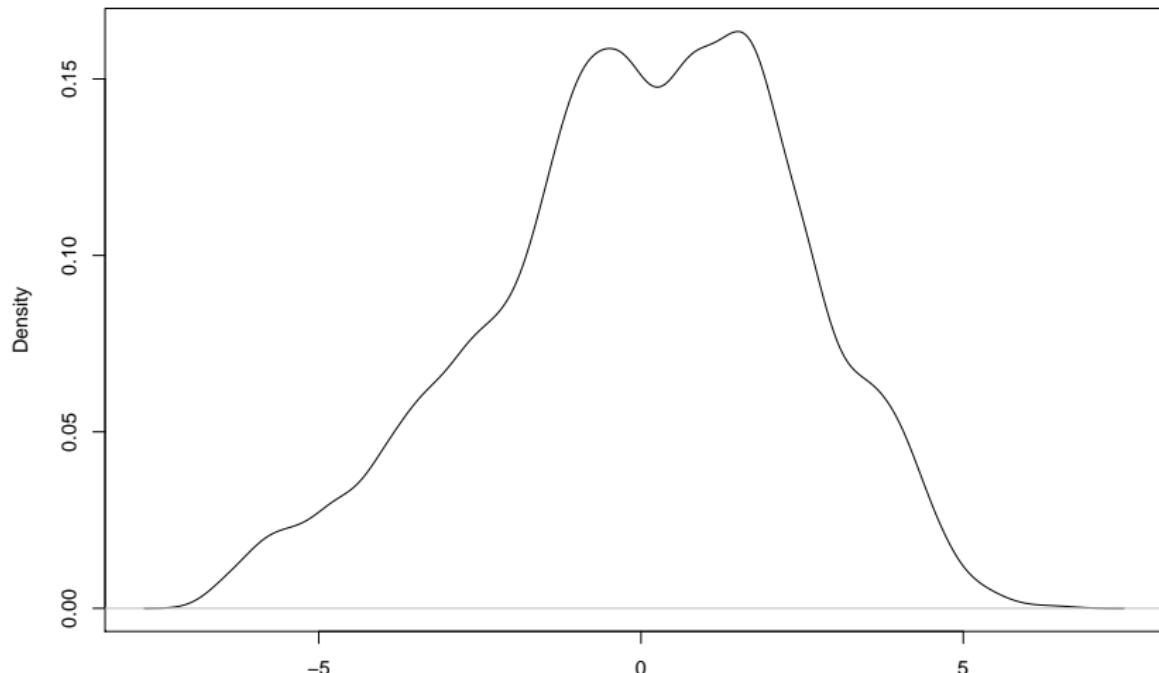
```
11$coefficients
```

```
## (Intercept)      happy income_dec
## 3.2077820    0.2470092   0.1207995
```

Accéder aux résidus

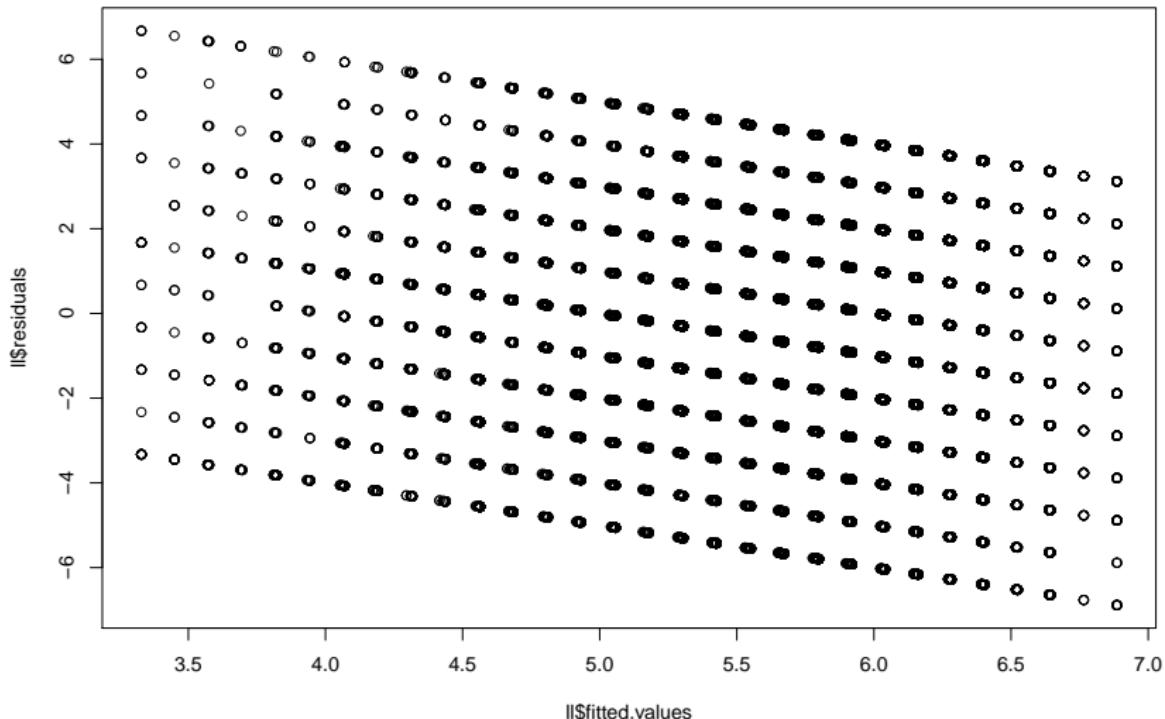
```
plot(density(l1$residuals))
```

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



Accéder aux résidus

```
plot(l1$fitted.values, l1$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")  
  
## Warning : Missing column names filled in : 'X1' [1]  
  
## Parsed with column specification :  
## cols(  
##   X1 = col_integer(),  
##   Name = col_character(),  
##   PClass = col_character(),  
##   Age = col_double(),  
##   Sex = col_character(),  
##   Survived = col_integer(),  
##   SexCode = col_integer()  
## )  
  
titanic <- mutate(titanic, Children = ifelse(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
##  All     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
## All      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
## All  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 = "b"  
##       data = titanic)  
##  
## Deviance Residuals :  
##      Min        1Q    Median        3Q       Max  
## -2.5991   -0.6638   -0.3799    0.7108    2.3085  
##  
## Coefficients :  
##                               Estimate Std. Error z value  
## (Intercept)          2.0848     0.2202  9.469  
## ChildrenEnfant      1.2582     0.2845  4.423
```

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 %
## (Intercept)     8.043257 4.656051 14.5035
## ChildrenEnfant 3.519043 1.707220  7.4303
## Sexmale        0.074435 0.043754  0.1228
## PClass2nd      0.411619 0.221909  0.7525
## PClass3rd      0.125013 0.065591  0.2302
##
##                      p
## (Intercept) < 2.2e-16 ***
## ChildrenEnfant 9.746e-06 ***
## Sexmale       < 2.2e-16 ***
## PClass2nd     0.0001744 ***
## PClass3rd     < 2.2e-16 ***
...  
...
```

Analyse géométrique de données

Packages

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

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)
d_acp <- select(d,
                  qfimedu, qfimlng, qfimchr,
                  qfimwht, qfimwsk, qfimcmt,
                  eduyrs, # Illus. quanti
                  gndr) # Illus quali
```

ACP

```
names(d_acp)

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

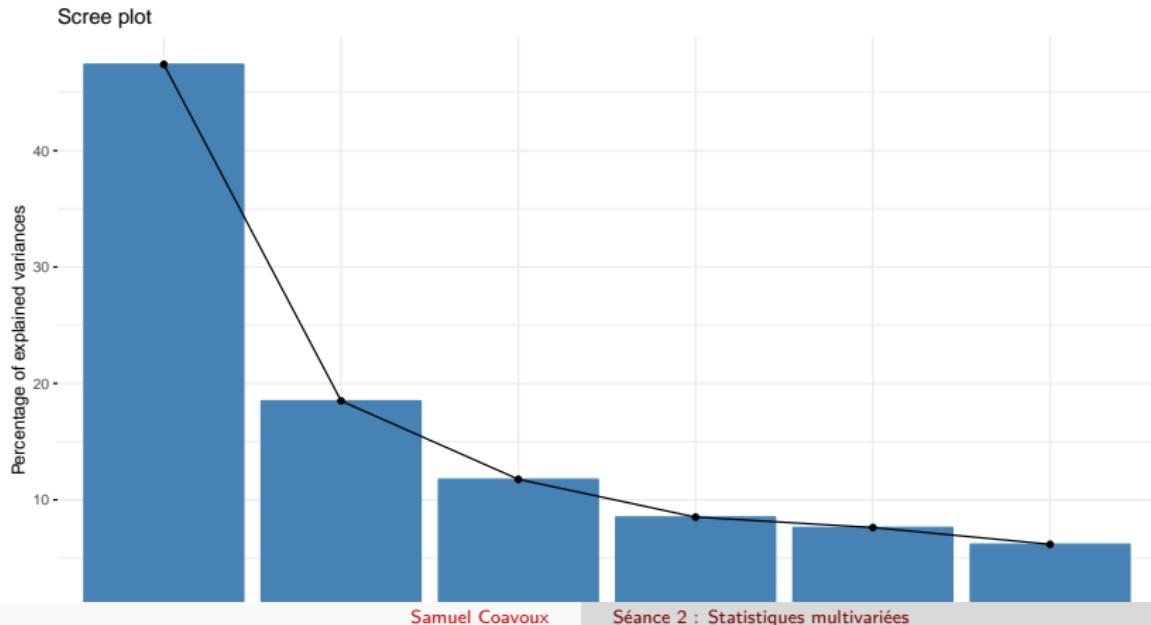
qfi_acp <- PCA(d_acp, quanti.sup = 7, quali.sup = 8,
                  graph = FALSE)

## 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

```
library(factoextra)
```

```
## Welcome ! Related Books : `Practical Guide To Cluster Analysis`  
fviz_eig(qfi_acp)
```



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  0.7308593 18.77852
## qfimlng  0.7192511 18.18674
## qfimchr  0.6137484 13.24264
## qfimwht  0.5922131 12.32962
## qfimwsk  0.8085636 22.98382
## qfimcmt  0.6417521 14.47866
```

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 -0.3405046 10.445303
## qfimlng -0.3361061 10.177187
## qfimchr  0.6194842 34.572877
## qfimwht  0.6524628 38.351866
## qfimwsk -0.2020234  3.676870
## qfimcmt -0.1755352  2.775897
```

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.

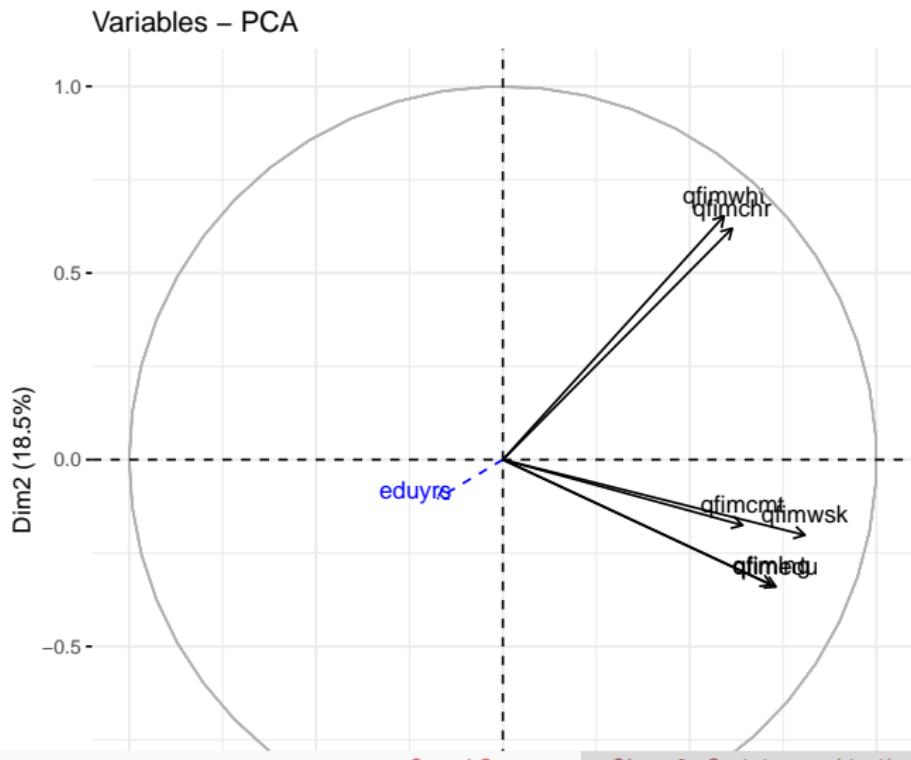
Par défaut, si on lance `PCA(d_acp, graph = TRUE)`, deux graphiques sont représentés : le nuage des individus, et le diagramme des variables, pour les deux premiers axes. C'est utile pour l'usage interactif ; dans les scripts, mieux vaut laisser `graph=FALSE` et produire ensuite explicitement les graphiques.

`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.

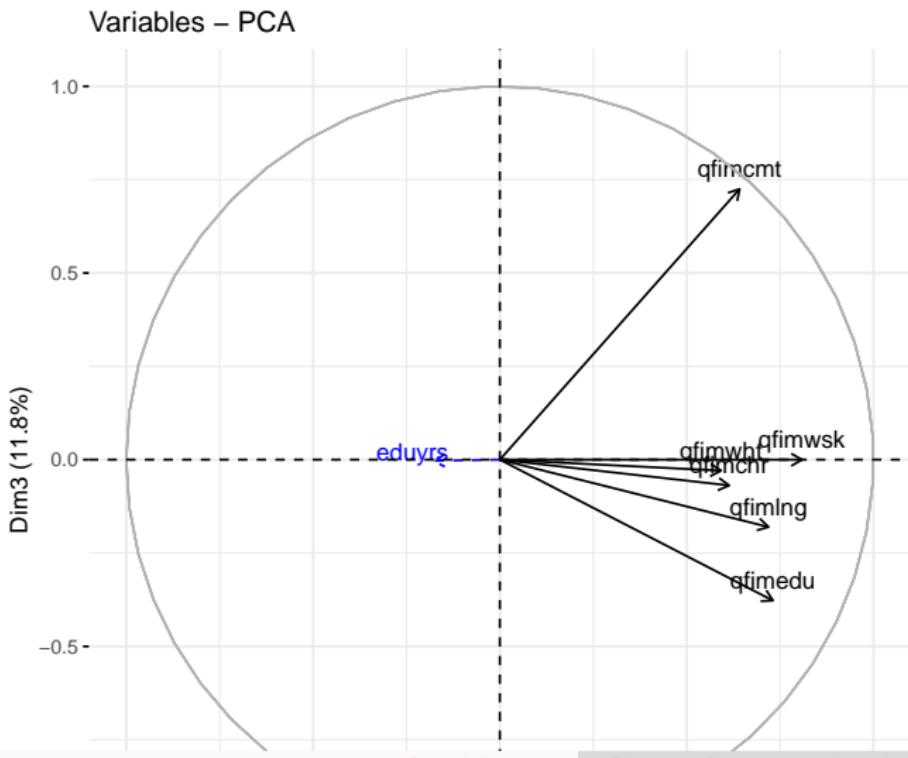
Variables, axes 1 et 2

```
fviz_pca_var(qfi_acp, axes = c(1, 2))
```



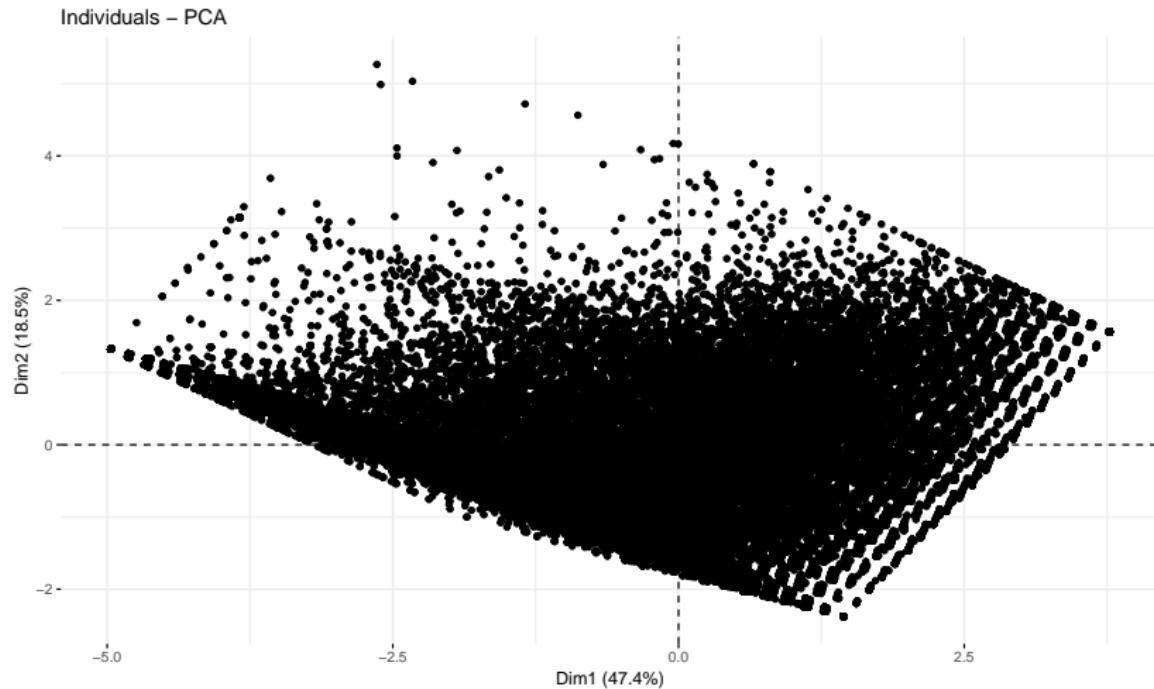
Variables, axes 1 et 3

```
fviz_pca_var(qfi_acp, axes = c(1, 3))
```



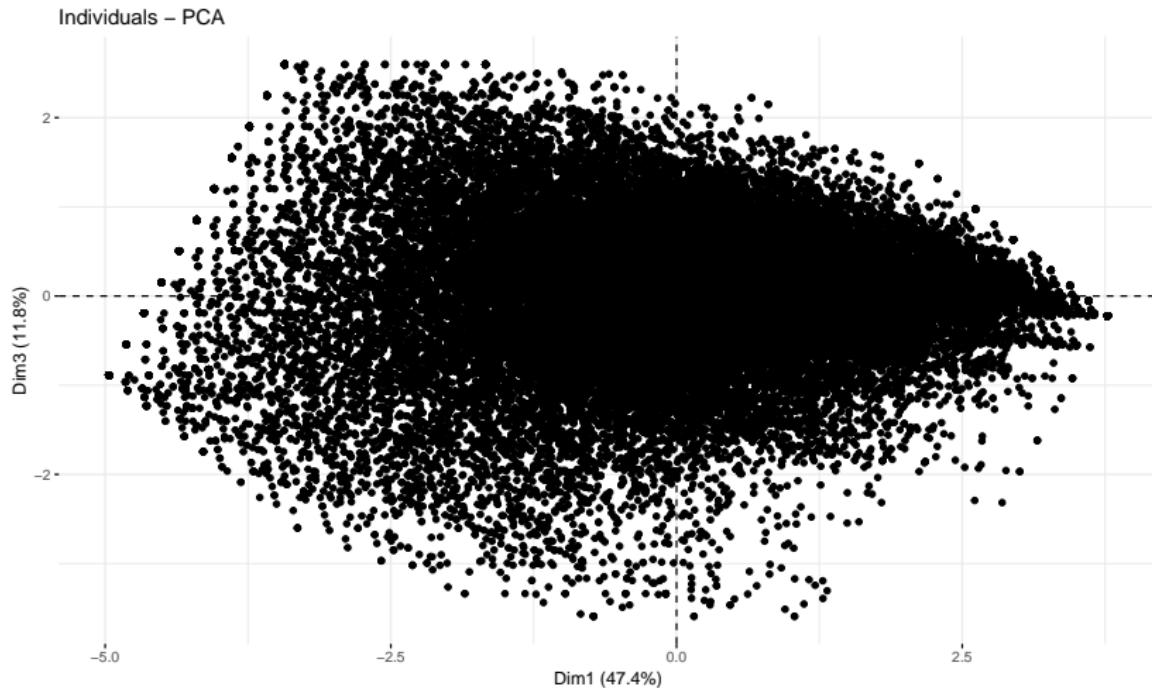
Individus, axes 1 et 2

```
fviz_pca_ind(qfi_acp, axes = c(1, 2), geom.ind = "point")
```



Individus, axes 1 et 3

```
fviz_pca_ind(qfi_acp, axes = c(1, 3), geom.ind = "point")
```



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(select(d_acp, qfimedu:qfimcmt))
complete_obs <- imputePCA(select(d_acp, qfimedu:qfimcmt), ncp =
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 ;
- 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.

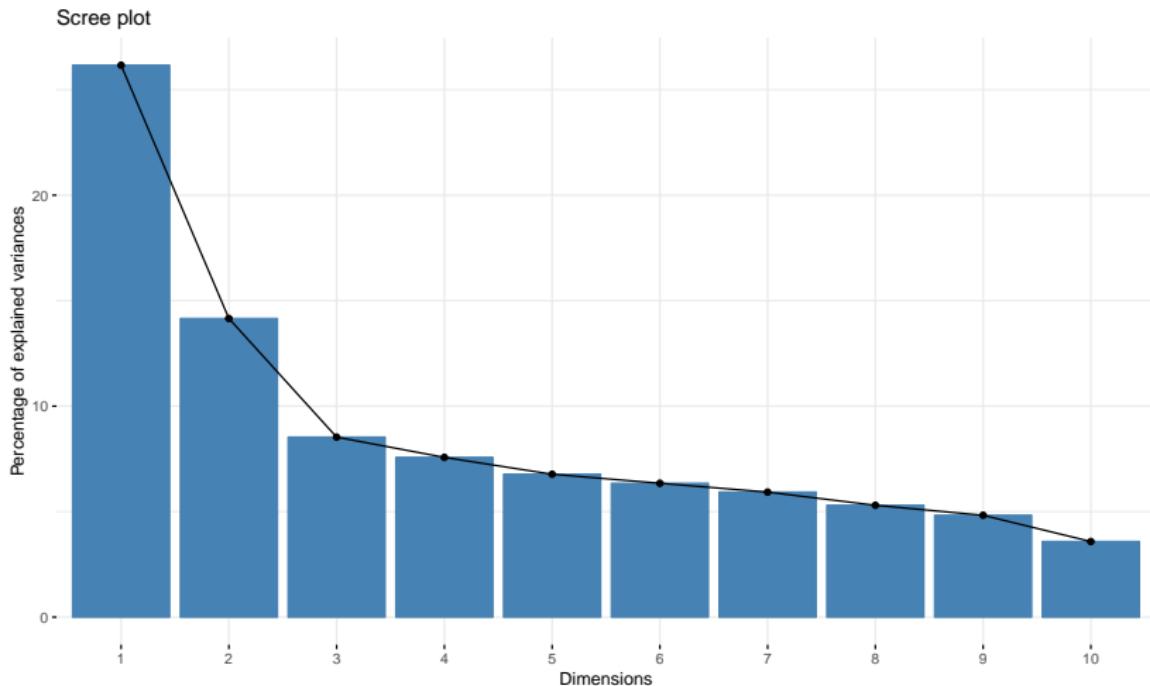
ACM

ACM

```
d_acm <- select(d, trstprl, trstlgl, trstplc,  
                  trstplt, trstprt, trstep,  
                  trstun, gndr, cntry)  
d_acm <- na.omit(d_acm)  
trust_acm <- MCA(d_acm, quali.sup = 8:9, graph = FALSE)
```

Eigenvalues

```
fviz_eig(trust_acm)
```



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  
## trstplt_High     1.37461179 7.07480106  
## trstpprt_Weak   -0.70603106 7.24510441  
## trstpprt_Average  0.58608258 3.16061294  
## trstpprt_High    1.00161652 3.34561300
```

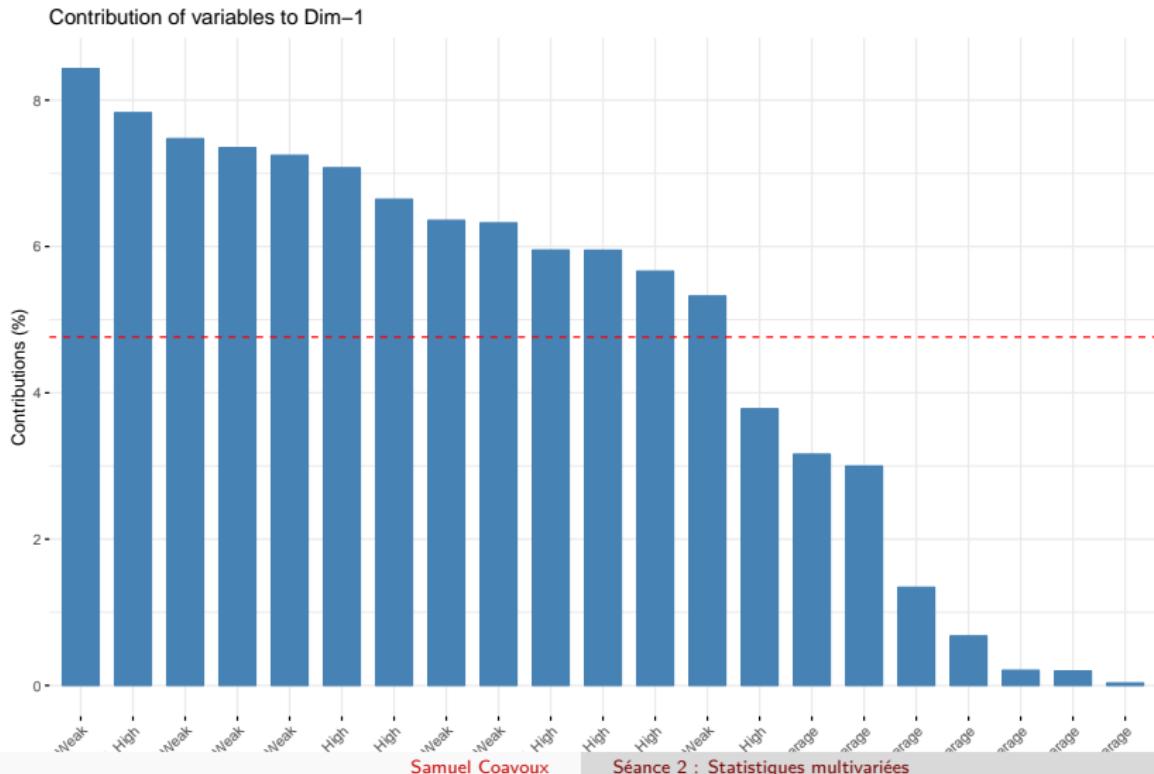
Contribution et coordonnées

```
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  
## trstplt_High     1.25822340 10.9538411  
## trstpprt_Weak    0.16310853  0.7145740  
## trstpprt_Average -0.76608788  9.9794747
```

Contributions axe 1

```
fviz_contrib(trust_acm, choice = "var", axes = 1)
```



Contributions axe 2

```
fviz_contrib(trust_acm, choice = "var", axes = 2)
```

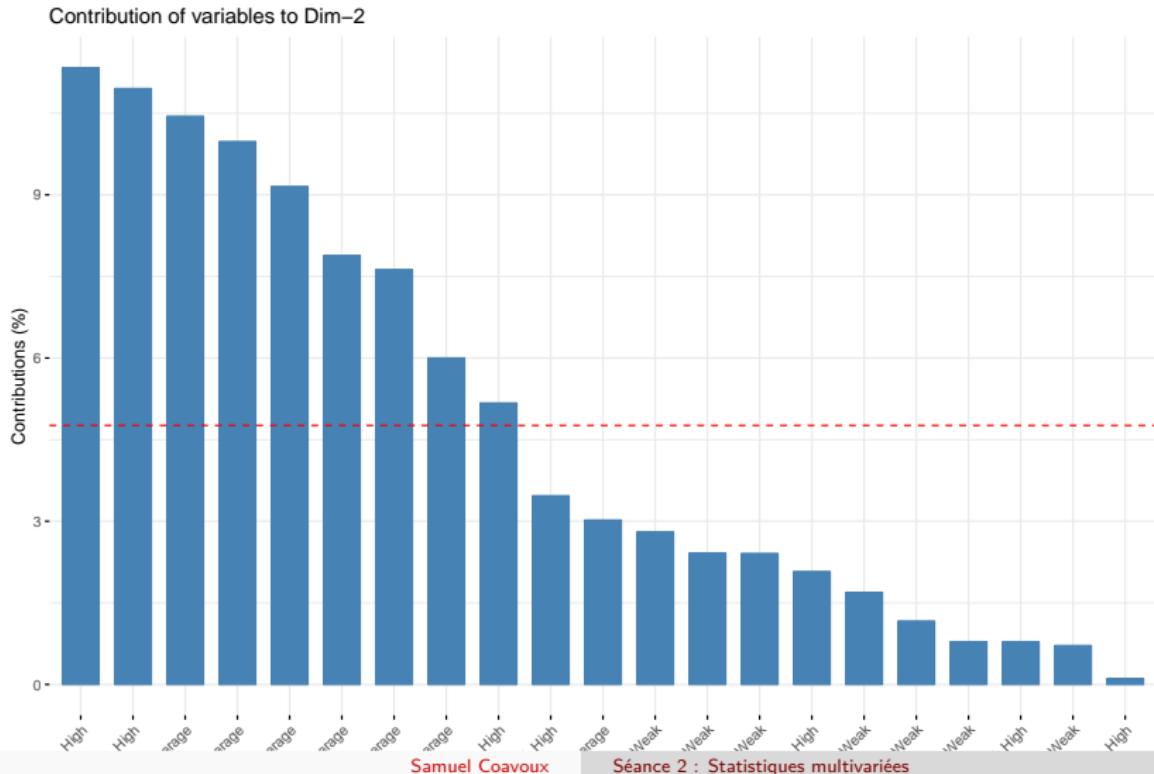
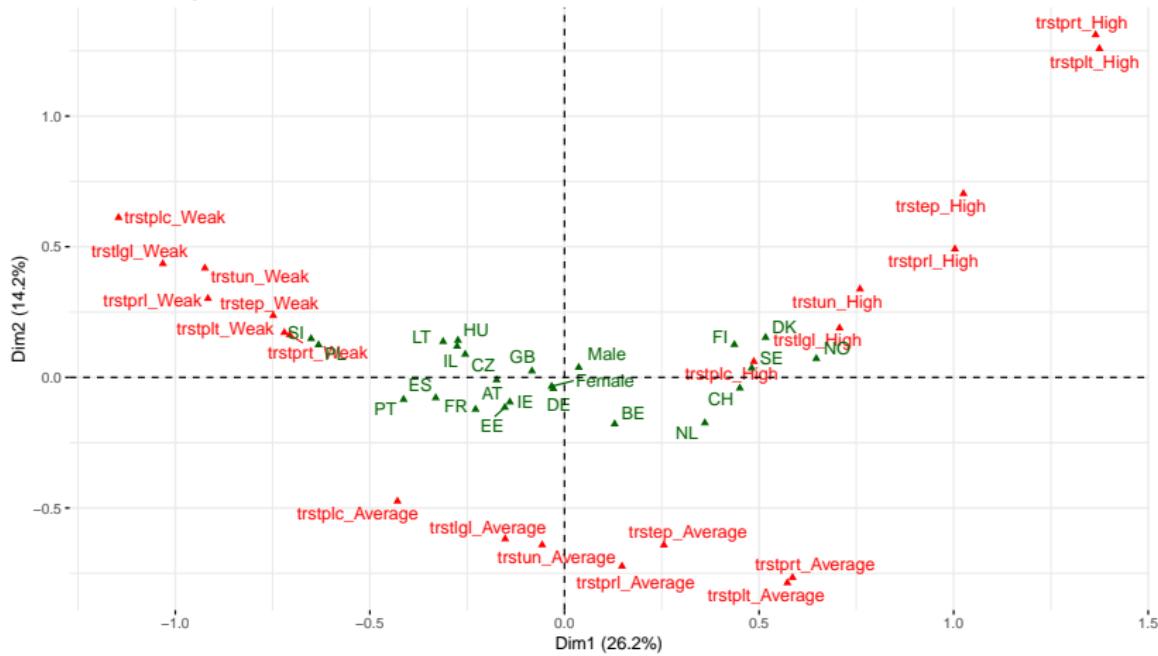


Diagramme des modalités

```
fviz_mca_var(trust_acm, repel=TRUE)
```

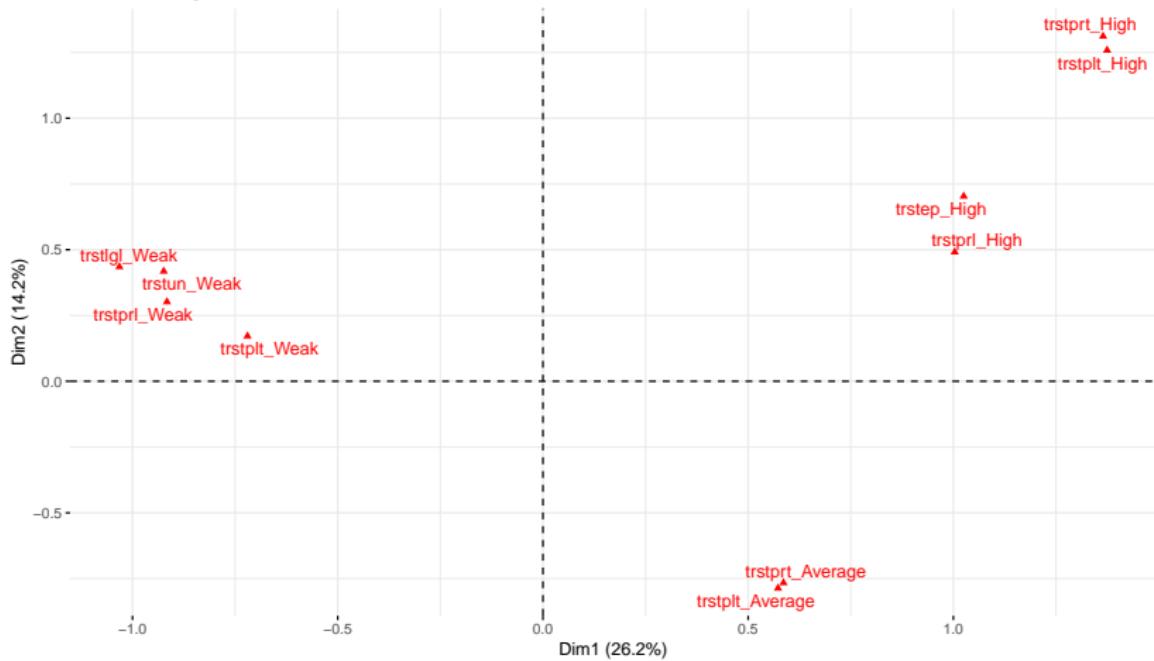
Variable categories – MCA



Sélection de modalités : 10 plus fortes contributions

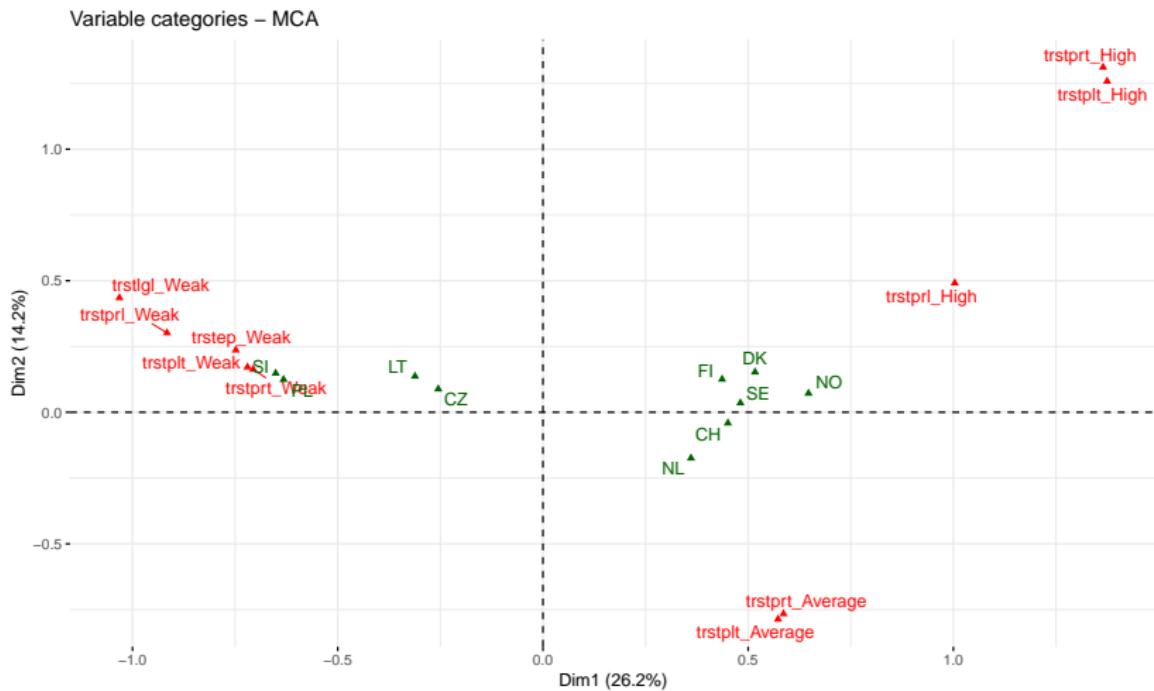
```
fviz_mca_var(trust_acm, repel=TRUE, select.var = list(contrib =
```

Variable categories – MCA



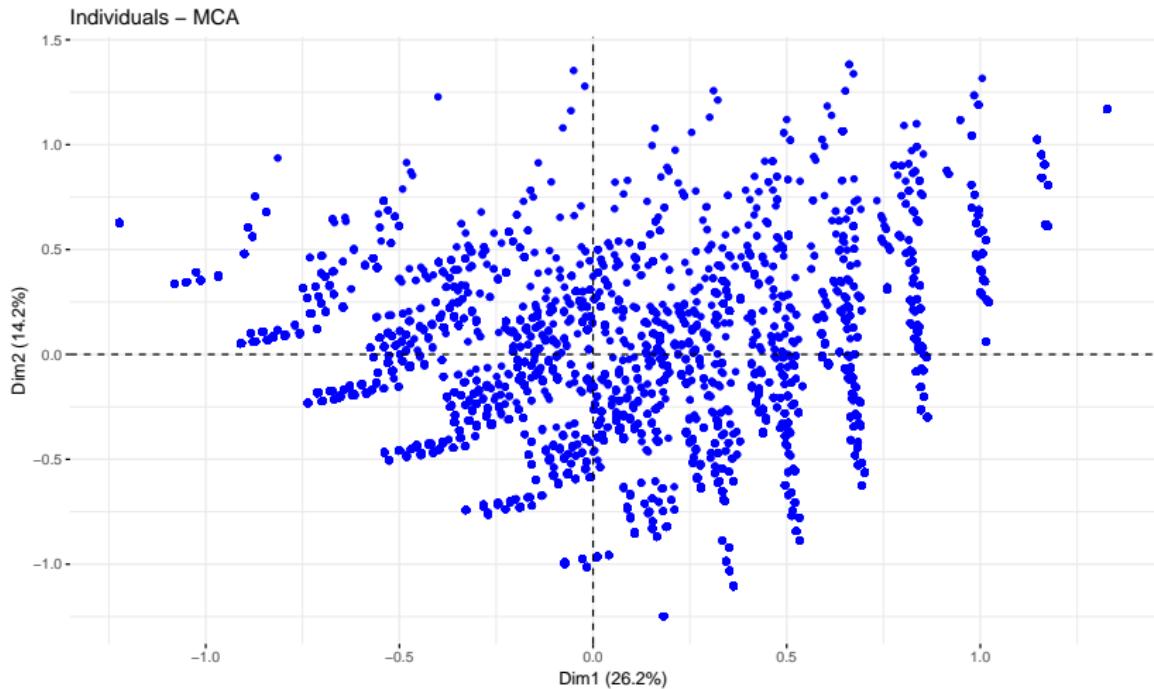
Sélection de modalités : 10 meilleures cos2

```
fviz_mca_var(trust_acm, repel=TRUE, select.var = list(cos2 = 10))
```



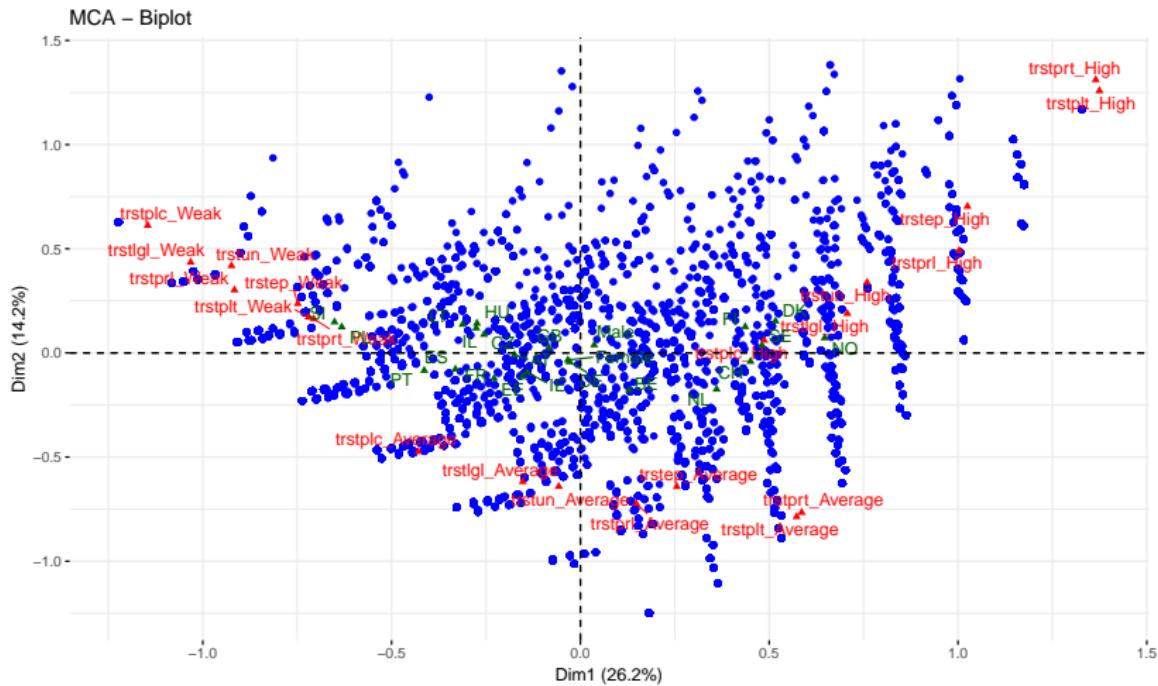
Nuage des individus

```
fviz_mca_ind(trust_acm, geom.ind = "point")
```



Nuage total

```
fviz_mca(trust_acm, geom.ind = "point", repel = TRUE)
```



Explor

Développé par Julien Barnier. Un package qui permet de produire les graphiques de façon interactive, avec ggplot en backend.

```
library(explor)
explor(trust_acm)
```

FactoInvestigate

Développé par François Husson (FactoMineR). Un package qui permet de réaliser automatiquement les interprétations classiques.

```
library(FactoInvestigate)  
Investigate(trust_acm)
```